UNIT: 3

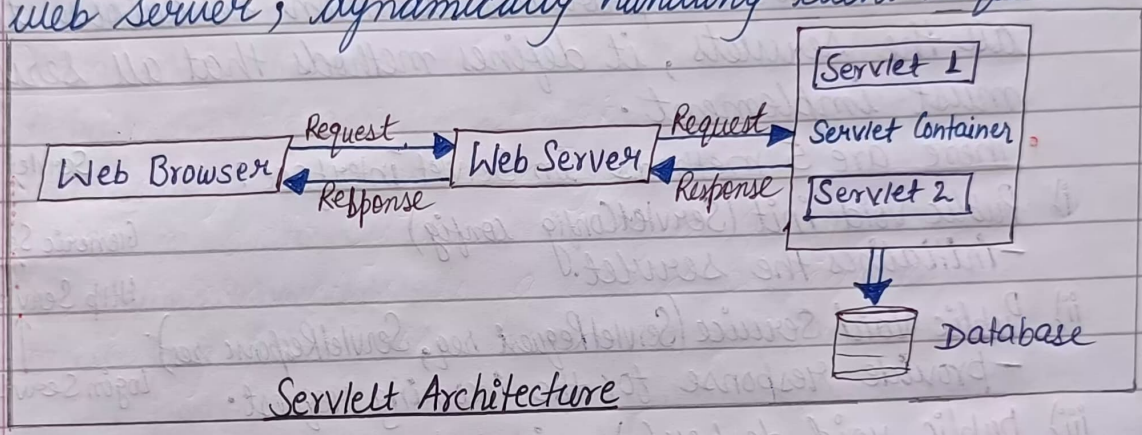
Dashrath Nandan

☆ Servlets

- A java class that runs on a web server and dynamically handles client requests. It is used to create a web application

- A web application is an app. accessible from the web. It is composed of web components like - Servlet, JSP; (Java Server Pages)

- A web component is a software entity that runs on a web server, dynamically handling client requests.



Servlet Architecture

- CGI (Common Gateway Interface) technology enable the web server to call an external program and pass HTTP request info. to the external program to process the request.

* Advantages of Servlet:

i) Better Performance : As it creates a thread for each request.
ii) Platform-Independent and Portability : bcz it uses Java language.
iii) Robust (iv) Secure

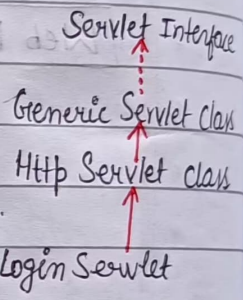
* Application of Servlet:

i) Read the explict data sent by the clients (browsers).
ii) Read the implict HTTP request data sent by the clients.
iii) Process the data and generate the results.
iv) Send the explict data (i.e. document) and implicit HTTP responses to the client.

* A web container is responsible for — managing the lifecycle of Servlets, mapping a URL to a particular Servlet and ensuring that the URL requester has the correct access-rights.

* Servlet Interface and Methods:
Servlets can be created using javax.servlet and javax.servlet.http packages.

- Servlet interface provides common behaviour to all the servlets, it defines methods that all servlet must implement.
- There are 5 methods in Servlet Interface:



i) Public Void init(ServletConfig config) — initializes the servlet.
ii) Public Void service(ServletRequest req, ServletResponse res) — provide response for the incoming request.
iii) public void destroy() — invoked only once, indicated Servlet is destroyed
iv) getServletConfig() — returns Object of ServletConfig.
v) public String getServletInfo() — returns info. about servlet, such as copyright, writer, version, etc.

* Generic Servlet Class
Generic Servlet class implements: Servlet, ServletConfig, and Serializable interfaces. It can handle any type of request so, it is protocol-independent.

Methods
public void init(.....): All 5 methods of Servlet Interface +
- public String getInitParameter(String name):
- public String getServletName(): return name of the servlet object.

* HttpServlet class :

HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace, etc.

↳ public void Service (ServletRequest req, ServletResponse res).
↳ protected void Service(" , "):
↳ protected void doGet(HttpServletRequest req, HttpServletResponse res):
↳ doPost / doHead / doOption / doTrace / doDelete.

➔ There are three ways to create a servlet –
i) implementing Servlet Interface ii) Extending Generic Servlet
iii) Extending HTTP Servlet.

Implementing Servlet Interface

```
import java.io.*;
import javax.servlet.*;
public class Dashrath implements Servlet{
ServletConfig config = null;
public void init(ServletConfig config)
{ this.config = config; }
public void service(SR req, SRes res)
  throws IOException, ServletException{
  res.setContentType("text/html");
  printWriter out = res.getWriter();
  out.print("<html><body>");
  out.print("<b>hello DNandan</b>");
  out.print("</body></html>"); }
public void destroy(){ SOP("Destroyed"); }
public ServletConfig getServletConfig(){
  return config; } }
```

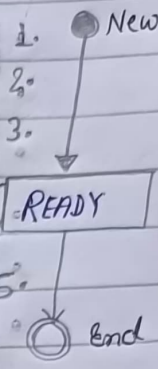
Initialized

Same code

Inheriting the Generic Servlet class

```
import java.io.*;
import javax.servlet.*;
public class Dashrath extends Generic
    Servlet{
public void Service(SR req, SRes res)
throws IOException, ServletException{
 res.setContentType("text/html");
printWriter out = res.getWriter();
out.print("<html><body>");
out.print("<b>Generic Servlet</b>");
out.print("</body></html>");
}
}
```

★ Life Cycle of a Servlet:



An HTTP servlet's life cycle is controlled by the web container where it is deployed.
There are three states of a Servlet: New, Ready, End.

1) Servlet class is Loaded: The Servlet class is loaded when first request for Servlet is received.
2) Servlet instance is created: The Servlet is in New state when servlet instance is created.
3) init method is invoked: Init method is used to initialize the servlet. After invoking init(), Servlet comes in Ready.
4) Service method is invoked: The web container calls the service method each time when request for Servlet is received.
5) Destroy method is invoked: It removes the Servlet instance from the service.

★ Servlets Request Interface:

The ServletRequest Interface is used to handle client request to access a servlet.

Methods:

- public Object getAttribute(String name): Returns the value of the named attribute.
- public int ContentLength: Returns the length of the request body.
- public String getContentType: Returns MIME type of request body.
- public String getParameter(String name)
- public String getProtocol: Returns the name and version of Protocol
- public int getRemotePort: Return the IP port of the client.

★ ServletResponse Interface:

The ServletResponse Interface defines an object to help a Servlet in sending a response to the client.
Servlet Mapping, Specifies the web container of which java Servlet should be invoked for a Url given by client.

Methods:

- public void flushBuffer: It forces the content in the buffer to be written to client.
- public int getBufferSize: Return the actual buffer size.
- public String getContentType:
- public PrintWriter getWriter: Return PrintWriter object to send characters text to client.
- public void reset: clears the buffer as well as status code.
- public void setBufferSize(int size):

Ex: Request to take username & password : (This is also ex. for HTTP Servlet)

// index.html

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

// web.xml

```
<servlet>
<servlet-name>test</servlet-name>
<servlet-class>Demo</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>test</servlet-name>
<...L-pattern>/test/url-pattern>
</servlet-mapping>
```

// Demo.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Demo extends HttpServlet
{protected void doPost(HttpServletReques
req, HttpSerRes res) throw ServletExcption
, IOExceptions
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
try{String username = req.getParameter("username
String password = req.getParameter("password");
pw.println("<h1> Hello "+"username"+"</h1>");
} finally{ pw.close}}}
```

☆ Creating Servlet Example in Eclipse (servlet with JDBC)

Eclipse is an open-source ide for developing JavaSE and Java EE (J2EE) application.

- You need to download the eclipse ide for Java EE devlopers.

1) Create the dynamic web project.

Click on File Menu -> New -> Project -> Web -> dynamic web project -> name -> finish

2) Create the Servlet in eclipse IDE: +Icon → explore the Java Resource → right click on Src → New → Servlet → name → uncheck all checkbox → except doGet() → next → Finish.

3) add jar file in eclipse IDE: right click on Project → Build Path → Configure Build Path → Libraries tab → External JARs button → select the servlet-api.jar file under tomcat/lib → OK.

4) Start the server and deploy the project
right click on Project → Run As → Run on Server → choose tomcat Server → next → addAll → finish.

☆ ServletConfig Interface

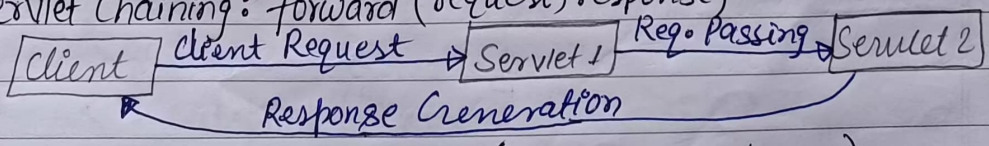
An object of ServletConfig is created by the web container for each servlet. The core advantage is that you don't need to edit the servlet file if information modified from web.xml file.

Methods:
- String getInitParameter (String name)
- Enumeration getInitParameterNames()
- String getServletName()

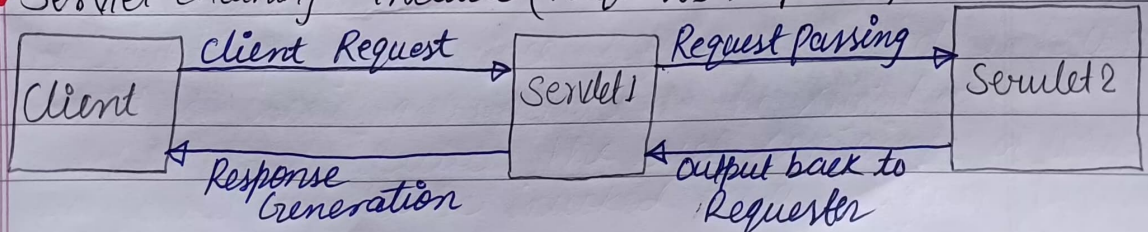
☆ Servlet Chaining

Used in order to FORWARD or INCLUDE a request from one servlet to another. RequestDispatcher interface provide two methods – RequestDispatcher.forward (request, respone).
RequestDispatcher.include (request, response).

- Servlet Chaining: forward (request, response)



- Servlet Chaining: include (request, response)



☆ JSP

JSP stands for Java Server Pages, a technology which allows the easy creation of Server side HTML pages. JSP consist of both HTML tags and JSP tags.

Features of JSP:

i) High performance and easy to maintain.
ii) Reduction in the length of code.
iii) Connection to Database is easier.
iv) Powered by Java - access to all Java APIs.

* JSP Elements

In JSP, elements can be divided into 4 different types -
1) Expression 2) Scriplets 3) Directives 4) Declarations

☆ JSP Scriptlet:

JSP Scripting elements provide the ability to insert java code inside the jsp.
There are three types of scripting elements:
- Scriptlet tag
- Expression tag
- declaration tag

*JSP Scriptlet tag: A scriptlet tag is used to execute java source code in JSP.

Syntax: <% java source code %>

Ex:-
```
<html>
<body>
<% out.print("Welcome to jsp"); %>
</body>
</html>
```

Note: Semicolon at the end of Scriptlet.

JSP expression tag:

We can use this tag to output any data on the generated page. These data are automatically converted to string and printed on the output stream.

Syntax: <%= Statement %> eg: <%= "Hello World" %>

JSP declaration tag:

Declaration tag is a block of java code for declaring class wide variables, methods and classes. The code written inside the jsp declaration tag is placed outside the Service() method of auto-generated Servlet.

Syntax: <%! field or method declaration %>

Ex:
```
<html>
  <head>
<title> Declaration tag Example </title>
  </head>
<body>
  <%! String name = "Dashrath" ; %>
  <%= "Name is :" + name %> <br>
  </body> </html>
```

JSP directives

The jsp directives are messages that tells the web container how to translate a JSP page into corresponding Servlet.

Type of directives:→ • Page directive , • include directive
• taglib directive.

Syntax of Directives:

<%@ directive name [attribute name = "value"...] %>

1) JSP page directive

The page directive defines attributes that apply to an entire JSP page.

Syntax: <%@ page attribute = "value" %>

* Attributes of JSP page directive –
  - import : <%@ page import = "java.io.*" %>
  - ContentType : <%@ page contentType = "text/html" %>
  - buffer : <%@ page buffer = "5kb/none" %>
  - language : <%@ page language = "java" %>
  - isELIgnorned : specify whether expression will evaluated or not.
  - pageEncoding • errorPage • isErrorPage.

2) JSP Include Directive

The include directive is used to include or copy content of one JSP page to another.

Syntax: <%@ include file = "value" %> here value is the JSP file name which needs to be included.

Note:- It can be used anywhere in the page.

Ex: <%@ include file = "myJSP.jsp" %>

3) Taglib Directive

Taglib directive allows user to use Custom tags in JSP. It helps you to declare custom tags in JSP page.

Syntax: <%@ taglib uri = "taglib URI" prefix = "tagPrefix" %>

Ex: <%@ taglib uri = "https://www.Sample.com/my custom lib" prefix = "demotag" %>

```
<html><body>
  <demotag:welcome/>
</body></html>
```

☆ JSTL

The JSP Standard Tag Library represent a set of tags to simplify the JSP development.

Advantages: i) Fast development. ii) Code Reusability
iii) No need to use scriptlet tag.

* JSTL Tags

JSTL mainly provides five types of tags:

1. Core tags: The JSTL core tag provides variable support, URL management, flow control, etc.

- Tags list:

↳ C:Out : It display the result of an expression.
↳ C:import : It Retrives relative or absolute URL and display the content
↳ C:Set : ,↳ C:remove ,↳ C:if → Conditional tag.
↳ C:Choose, C:When, C:Otherwise → Simply conditional tags.
↳ C:forEach → Basic iteration tag, ↳ C:param : adds parameter.
↳ C:redirect ↳ C:Url → Create a Url.

2. JSTL Function Tags:

The JSTL function provides a number of standard functions, most of these functions are common string manipulation functions.

- Tags list:

- fn:Contains() → Test if an input String containing the substring.
- fn:ContainsIgnoreCase() → • fn:endsWith()
- fn:escapeXml() → escapes char. that is interpreted as XML markup.
- fn: indexOf() • fn: trim() • fn:StartWith()
- fn: toLowerCase() • fn: toUpperCase()

3. JSTL Formatting tags:

It provides support for message formatting, number and date formating.

<%@ taglib Uri = "http://java.sun.com/jsp/jstl/fmt" prefix = "fmt" %>

Taglist:

fmt: parseNumber; fmt: timeZone ; fmt: parseDate ;
fmt: setTimeZone ; fmt: Bundle ; fmt: message
fmt: formate date.

4. JSTL XML Tags:

These tags are used for prouiding a JSP-centric way of manipulating and creating XML document.

<%@ taglib uri = "http://java.sun.com/jsp/jstl/xml" prefix = "x" %>

Taglist:

X: out ; X: parse ; X: set ; X: Choose ; X: when
X: Otherwise ; x: if ; X: transform ; X: param.

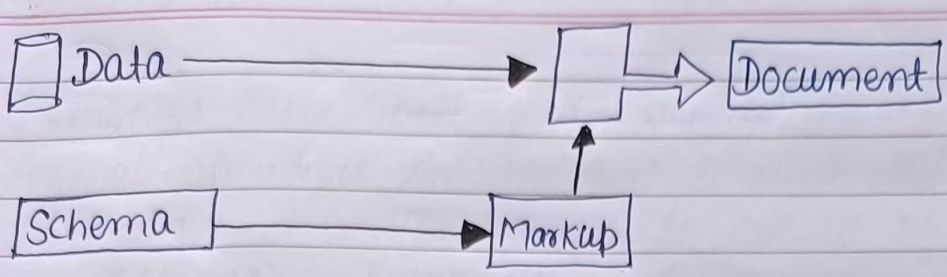
5. JSTL SQL Tags

It provide SQL support.

<%@ taglib uri = "http://java.sun.com/jsp/jstl/sql" prefix "sql" %>

Taglist:

- Sql: SetDataSource → Used for creating simple data source.
- Sql: query → used for executing SQL query.
- Sql: update →
- Sql: param
- Sql: dateParam
- Sql: transaction.



Document, is a combination of data and markup.
Markup, describes the data in the document and how it should be interpreted.

☆ XML

Extensible Markup Language. A Standard for data exchange. Designed to describe data.

- It is used to - Separate data from HTML, exchange data, store data, Share data.

Advantages:

i) Human readable : XML uses simple text-based format.
ii) Allow validation :
iii) Extensible : Custom tags can be created & used easily.
iv) Technology agnostic : XML is technology independent.

Disadvantages :

i) Redundant Syntax.
ii) Verbose : XML file size increases costs.

Example :

```
<?xml version = "1.0" ?>   ← Prologue
<Person>                   ← Root Element
<Name> Dashrath </Name>    }
<Age> 21 </Age>            }  Document
</person>
```

✱ XML Building Blocks :

1) Prolog : A part of XML document that precedes XML data.
- It includes - A declaration, An optional DTD.

ii) Elements: Basic unit of an XML document. It is a logical structure in XML that is delimited by a start tag and an end tag. It consist of 3 parts -

<Name> Dashrath </Name>

Start Tag — Content — End Tag

- Start and End Tags should match and is case sensitive.

iii) Attributes: Provides additional information. Specified in the start tag of the element, Have a key-value pair. Attribute can have only one value.

<Flower COLOR = "RED"> ROSE </Flower>

Attribute Name — Attribute Value

iv) Well-formed XML Document

An XML document is said to be well-formed if it follows basic syntax rules specified for XML by W3C.

- Must have only one root element. Every element must have a closing, tag Elements must be properly nested.

★ XML Tree Structure

An XML document has a self-descriptive structure. It forms a tree structure called XML tree.

Note: DOM parser represents the XML Document in Tree Structure.

XML Tree Rules → Used to figure out relationship of elements.

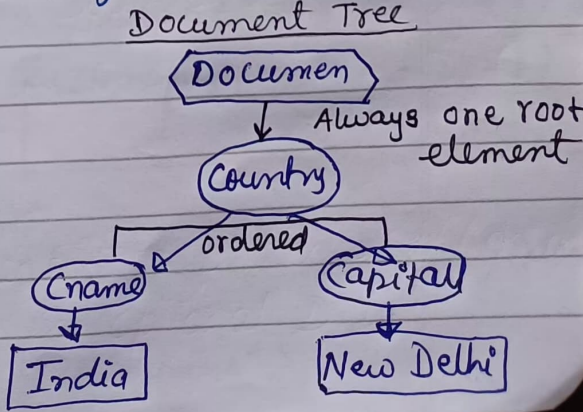
i) Descendants: If element 'A' is contained by 'B', then A is descendant of B.

ii) Ancestors:

XML Document

```
<country>
 <cname> India </cname>
 <capital> New Delhi </capital>
</country>
```

Document Tree



* Valid XML Document:

An XML document is valid if it - is well formed, Declares a DTD (Document Type Defination), DTD and XML schemas provides descriptions of document structures.

☆ DTD (Document Type Defination)

- DTD defines the legal building block of an XML document.
- It is a means to validate XML document.
- A <!DOCTYPE> element is used to create a DTD.

The element can take different forms -

```
<!DOCTYPE rootname [DTD]>
<!DOCTYPE rootname SYSTEM URI>
<!DOCTYPE rootname PUBLIC identifier URI>
```

* Types Of DTD:

1) **Internal DTD:** Also known as internal subset.

```
<!DOCTYPE Rootname [element declaration]>
```

2) **External DTD:** Also known as external subset. declaration lies in the external document.

```
<!DOCTYPE Book SYSTEM "output.dtd">
```

Example:

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd"> // external DTD file
<employee>
<first name> Dashrath </first name>
<last name> Nandan </last name>
</employee>
```

employee.dtd

```
<!Element employee (first name, last name)>
<!Element first name (#PCDATA)>
<!Element last name (#PCDATA)>
```

* XML DTD with entity declaration:

An entity has three parts: An ampersand (&), An entity name, A Semicolon (;).

```
<!ENTITY entity name "entity value">
```

* **CDATA**: Unparsed Character data, contains the text which is not parsed further in an XML document. Tags insides the CDATA text are not treated as markup and entities will not expanded.

* **PCDATA**: Parsed Character Data. It is the text parsed by a parser.

Example:

CDATA

```
<? xml version = "1.0" ?>
<! DOCTYPE employee SYSTEM "employee.dtd">
<employee>
<![CDATA[
  <firstname> Dashrath </firstname>
<lastname> Nandan </lastname>
]]>
</employee>
```

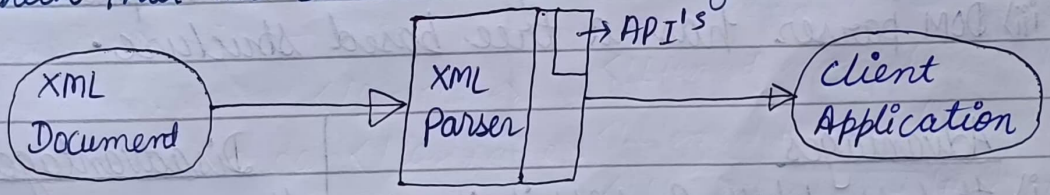
PCDATA

same

```
<employee>
<firstname> Dashrath </firstname>
<lastname> Nandan </lastname>
</employee>
```

XML Parsers

XML Parser is a software library or package that provides interfaces for client application to work with an XML document. XML Parser validate the document and check that the document is well formatted.



* **Types of XML Parser:**

i) DOM
ii) SAX

1) SAX (Simple API for XML)

A SAX Parser implements. This API is an event based and less intuitive.

Features:

i) It doesn't create any internal structure.
ii) Client doesn't know what methods to call, they just override the methods of API and place their code.
iii) It is an event based parser.

Advantages	Disadvantages
i) It is simple & memory efficient.	i) Event-based so less intuitive.
ii) It is fast and work for huge document.	ii) Client never know full info.

2) DOM (Document Object Model)

A DOM document is an object which contains all the information of an XML document. It is composed like a tree structure. DOM Parser implements DOM API.

Features:

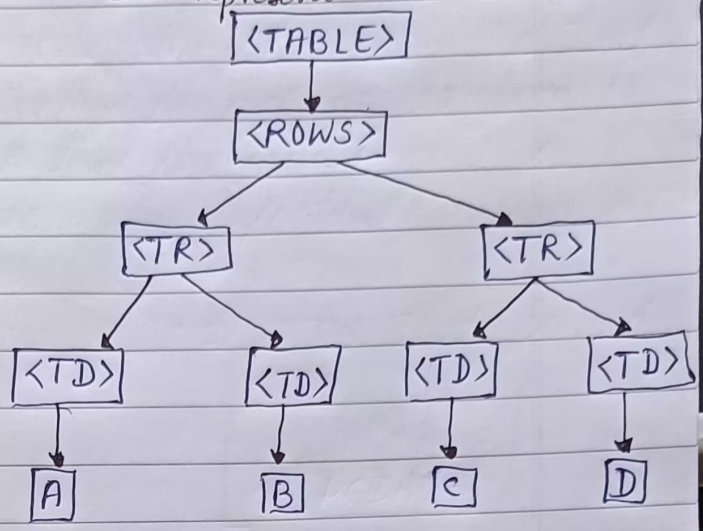
i) It creates an internal structure in memory.
ii) It defines standard way to access and manipulate XML docs.
iii) DOM parser has a tree based structure.

Advantages	Disadvantages
i) It support both Read & Write Operations.	• It is memory inefficient.
ii) API is very simple to use.	• It is slower.
iii) It is preferred when random access to widely separated parts of a document is required.	

Example:

```
<TABLE>
<ROWS>
<TR>
<TD> A </TD>
<TD> B </TD>
</TR>
<TR>
<TD> C </TD>
<TD> D </TD>
</TR>
</ROWS>
</TABLE>
```

DOM represent table like this –

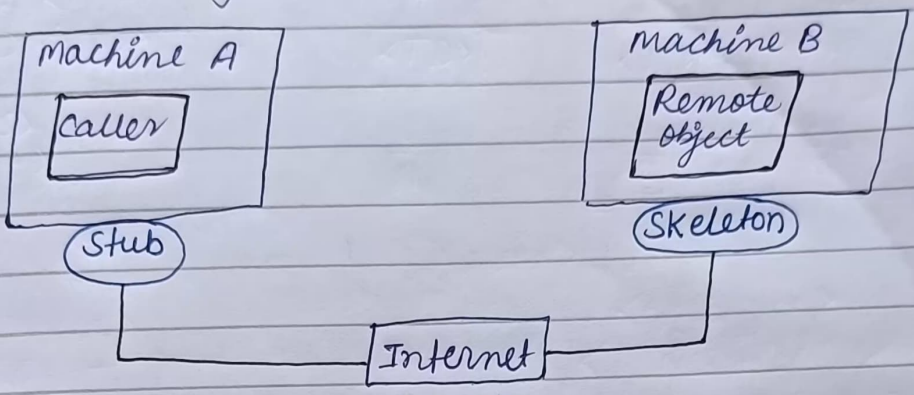


★	SOAP	REST
i)	Simple Object Access Protocol.	→ REpresentational State Transfer.
ii)	SOAP is a protocol.	→ REST is an architectural style.
iii)	It uses service interface to expose business logic.	→ It uses URI to expose business logic.
iv)	It defines its own Security.	→ It inherits security measures.
v)	SOAP permits XML data format only.	→ REST permits different data formats - Plain Text, HTML, JSON, etc.
vi)	Less preferred.	→ More preferred than SOAP.
vii)	JAX-WS is the java API for SOAP web service.	→ JAX-RS is the java API for RESTful web services. *Some imp. JAX-RS annotations: @Path, @GET, @PUT, @POST, @DELETE, @Produce, @consumes, @PathParam.

★ RMI (Remote Method Invocation)

The RMI is an API that provides a mechanism to create distributed application in java.

- The RMI provides remote communication between the applications using two objects Stub and Skeleton.



* **Stub:** The stub is an object, acts as a gateway for the client side. When the caller invokes method on the stub object, it does —
  1. it initiates a connection with remote Virtual machine (JVM).
  2. it writes and transmits (marshals) the parameter to JVM.
  3. it waits for the results.
  4. It reads (Unmarshals) the return value or exception
  5. It finally, return the value to the caller.

* **Skeleton:** The Skeleton is an object, acts as a gateway for the server side. When it receive requests —
  1. it reads the parameter for the remote method.
  2. it invokes the method on the actual remote object,
  3. it writes and transmits (marshals) the result to caller.

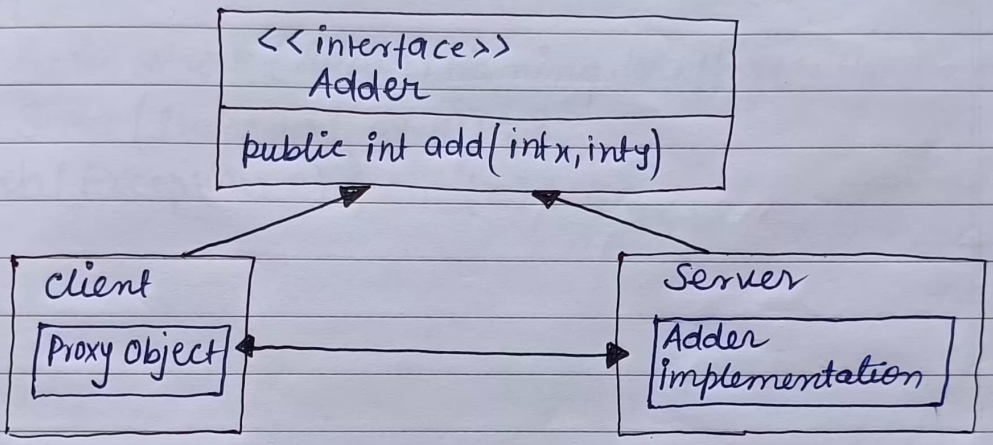
* **Requirements for distributed application —**

i) The application need to locate the remote method
ii) It need to provide the commun^c with remote object.
iii) The app need to load the class defination for object.
RMI has all these features.

* Java RMI Example:

There are 6 steps to write the RMI program :-

Step 1: Create the remote interface.
Step 2: Provide the implementation of the remote interface.
3: Compile the implementation class and create Stub and skeleton object using the rmic tool.
4: Start the registry service by rmiregistry tool.
5: Create and start the remote application.
6: Create and start the client application.



Codes step by step :

Step 1

```
import java.rmi.*; import java.rmi.server.*;
public interface Adder extends Remote{
public int add(int x, int y) throws RemoteException;}
```

Step 2

```
public class AdderRemote extends UnicastRemoteObject
  implements Adder{
  AdderRemote() throws RemoteException{ super(); }
  public int add(int x, int y){ return x+y ;}
```

Step 3: rmic AdderRemote

Step 4: rmiregistry 5000

Step5:
```
public class MyServer{
psvm (String[] args){
try{
    Adder stub = new AdderRemote;
    Naming.rebind("rmi://localhost5000/sonoo", stub);}
catch(Exception e){s.out(e);}}}
```

Step6:
```
public class Myclient{
psvm(String[] args){
try{
    Adder stub=(Adder)Naming.lookup("rmi://localhost5000/sonoo");
    S.out(stub.add(34,4));}
Catch(Exception e){sop(e);}}}
```