

## ★ What is Distributed software development ?

Distributed software development involves creating applications that run on multiple interconnected computers.

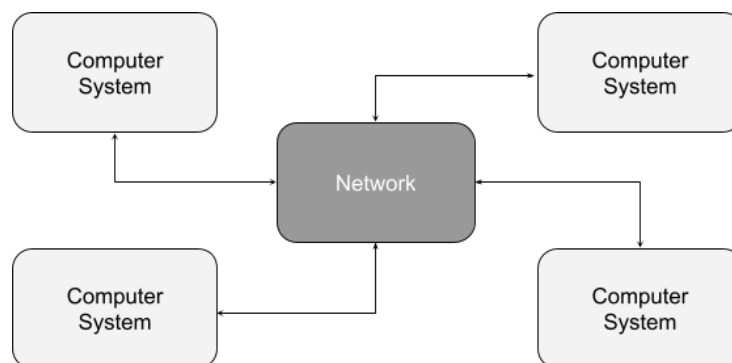
- **Modularity** : Break down the system into manageable modules
- **Loose Coupling** : Reduce dependencies between modules
- **Scalability** : Ability to handle increasing workloads
- **Reliability** : System remains operational under varying conditions
- **Security** : Protect data and resources from unauthorized access

## ★ What is a distributed application / system?

A distributed application is a software application that consists of multiple components running on different computers connected by a network.

- Web applications
- Multiplayer online games
- Distributed databases
- Internet of Things (IoT) applications

Examples of distributed systems include cloud computing platforms, peer-to-peer networks, and distributed databases.



## ★ Role of Java in Enterprise Development

Java provides libraries and frameworks (e.g., Apache Kafka) for building distributed systems.

### Benefits of building enterprise applications with Java

- **Platform Independence** : Write once, run anywhere
- **Robustness** : Strong memory management, garbage collection
- **Security** : Built in security features
- **Scalability** : Ability to handle large scale applications
- **Performance** : Optimized virtual machine (JVM)

## ★ Why you should choose java for enterprise applications.

Cross-Platform Compatibility, Simple to Use, Stable Language, Speed and performance, Powerful Development Tool, Availability of Libraries, Security, Rich API

## ★ Java in Enterprise Development

1. **Web Applications** : Servlets, JSP, Spring Framework
2. **Enterprise Integration** : JMS, JNDI, EJB
3. **Database Connectivity** : JDBC, Hibernate
4. **Messaging** : JMS (Java Message)
5. **Web Services** : SOAP, RESTful services

## ★ Enterprise Application Architecture

-

### Components

- **Presentation Layer** : User interface (
- **Business Logic Layer** : Application logic
- **Data Access Layer** : Database interactions
- **Integration Layer** : Connects with external systems

### Java Technologies

- **Servlets and JSP** : Presentation layer
- **Spring Framework** : Business logic and integration
- **Hibernate** : Data access layer
- **JMS** : Messaging between components

Example:

### Online Retail System (Web Site)

#### Components

- **Presentation Layer** : Web interface for customers to browse and purchase
- **Business Logic Layer** : Manages shopping cart, order processing, and inventory
- **Data Access Layer** : Connects to database to store and retrieve product and customer information.
- **Integration Layer** : Interfaces with payment gateways, shipping services, and external

#### Java Technologies Used

- **Servlets and JSP** : Presentation layer, handling user interactions and displaying.
- **Spring Framework** : Business logic, providing services such as shopping cart management and order processing.
- **Hibernate** : Data access layer, interacting with the database to store and retrieve product and customer data.
- **JMS** : Messaging between components for asynchronous processing and communication with external systems.

## ★ What is a database ?

A database is a structured collection of data that can be easily accessed, managed, and updated.

### 1. Relational Databases:

These databases organize data into tables with rows and columns, where each row represents a record and each column represents an attribute.

Examples - MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server.

This short note making resources are, google search engine, chat gpt and lecture slides  
[WhatsApp group](#) (click on this/new syllabus notes only)

Credit - [Kanishka viraj](#)

## 2. NoSQL Databases:

These databases are designed to store and manage unstructured, semi-structured, or structured data.

They are often used for big data and real-time web applications.

Examples - MongoDB, Cassandra and Redis.

### ★ What is JDBC (Java Database Connectivity) ?

JDBC (Java Database Connectivity) is an API that enables Java applications to interact with databases.

#### JDBC Architecture

- **Application:** The Java application that interacts with the database.
- **JDBC API:** Defines the interfaces and classes for JDBC programming.
- **Driver Manager:** Manages the JDBC drivers and provides methods for establishing connections.
- **JDBC Driver:** The driver implements the JDBC interfaces to communicate with the database.

### ★ JDBC Connection Code

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class MySQLExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/DATABASE";
        String username = "USERNAME";
        String password = "PASSWORD";

        try {
            Connection conn = DriverManager.getConnection (url, username, password);
            System.out.println ("Connected to MySQL database!");
            conn.close();
        }
        catch (SQLException e) {
            System.out.println("Failed to connect to MySQL database!");
            e.printStackTrace();
        }
    }
}
```

## ★ Try - Catch

In Java, the try and catch blocks are used for exception handling.

Exceptions are events that occur during the execution of a program that disrupt the normal flow of instructions.

They can occur for various reasons, such as invalid input, file not found, or database connection issues.

## ★ DBMS vs RDBMS (Relational Database Management System)

DBMS	RDBMS
DBMS stores data as a file.	RDBMS stores data in tabular form
Normalization is not present.	Normalization is present.
It deals with a small quantity of data.	It deals with large amounts of data.
Data redundancy is common in this model.	Keys and indexes do not allow Data redundancy.
It supports single users.	It supports multiple users.
Data elements need to be accessed individually.	Multiple data elements can be accessed at the same time.
No relationship between data.	Data is stored in the form of tables which are related to each other.

## ★ What is a web based Application?

Web based applications are software programs accessed through web browsers, providing functionality and services over the internet.

Examples - email clients, social media platforms, online shopping portals, and productivity tools.

## ★ Client side vs. Server side Development

- 1. Client-side development:** Programming executed on the user's device (browser), using HTML, CSS, and JavaScript to create interactive interfaces.
- 2. Server-side development:** Programming executed on the web server, managing data, business logic, and communication with databases.

Client-side	Server-side
It runs on the user's computer	It runs on the web server
It does not provide security for data.	It provides more security for data.
Source code is visible to the user.	Source code is not visible to the user
HTML, CSS, and javascript are used.	PHP, Python, Java, Ruby are used.
No need for interaction with the server.	It is all about interacting with the servers.

## ★ What is State in Web Applications?

State refers to the current condition or data of a web application, including user inputs, session details, and application state

e.g., logged in status, shopping cart contents

## ★ Types of State Management

### 1. Client Side State Management:

Involves storing and managing state data on the client's browser.

Techniques include - cookies, local storage, session storage, and client side frameworks/libraries (e.g., Redux for React).

### 2. Server Side State Management:

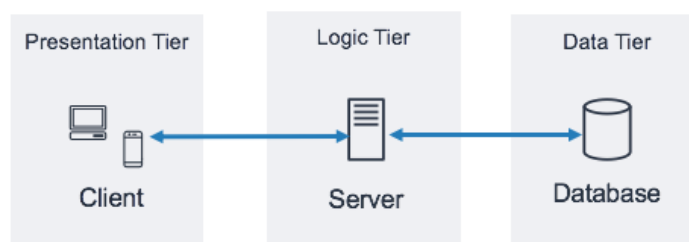
Involves storing state data on the server, often in databases or server memory.

Techniques include - sessions, databases, and server side frameworks/libraries (e.g., Express.js for Node.js).

## ★ Types of client server architecture

- 1-tier architecture
- 2-tier architecture
- 3-tier architecture

presentation layer - for user interaction ( Front-end / UI Layer)  
 Business Logic Layer / application layer - for processing and logic. ( Middle layer )  
 database layer - for data storage. ( Back-end )



- N-tier architecture ( N-tier architecture is multilayered client-server )

## ★ What are Dynamic Web Models ?

Dynamic web models are architectures and patterns that enable dynamic content generation, user interactions, and real-time updates in web applications.

Dynamic Website is a website containing data that can be mutable or changeable.

It uses client-side or server scripting to generate mutable content. Like a static website, it also contains HTML data.

These models enhance user experience, data presentation, and responsiveness by leveraging technologies like,

**AJAX, WebSocket, and server side rendering.**

### 1. AJAX ( Asynchronous JavaScript and XML )

Allows asynchronous data retrieval and updating parts of a web page without full page reloads.

AJAX (Asynchronous JavaScript and XML) enables web pages to update parts of the content asynchronously by fetching data from the server without reloading the entire page.

### 2. WebSockets

WebSockets enable real-time, two-way communication between the client and server, allowing instant data exchange and event-driven interactions.

### 3. Single Page Applications (SPAs)

Loads a single HTML page and dynamically updates content using JavaScript frameworks / libraries (e.g., React, Angular, Vue.js).

## ★ Advantages of dynamic Web Models

1. It provides real-time data.
2. It's easy to modify or update data.
3. It provides a user-friendly interactive interface for users.
4. provide interactive user interface
5. It provides a better user experience.

## ★ Disadvantages of dynamic web Models:

1. These types of websites are complex.
2. These are more expensive to develop.
3. high production costs.
4. Slow to load content.
5. Low speed compared to a static website
6. Hosting of these websites is also costlier.
7. Clients will require a skilled programmer to build a dynamic website.
8. Hosting a website is costly as compared to a dynamic website.

## ★ MVC Architecture ( Model-View-Controller )

This pattern is the basis of interaction management in many web-based systems.

MVC (Model-View-Controller) is a pattern in software design commonly used to implement user interfaces, data, and controlling logic.

## ★ What is XML ?

XML stands for Extensible Markup Language. It is a text based markup language derived from Standard Generalized Markup Language (SGML).

- XML was designed to store and transport data.
- XML was designed to be both human and machine readable.
- A language for describing data
- A tool for building new data description languages
- Open to define in any way you want
- A markup language. That means that “marks” or tags are used to identify the data elements. Data and tags are generally stored as plain text.

## Difference between HTML and XML

HTML	XML
HTML stands for Hyper Text Markup Language.	XML stands for Extensible Markup Language.
HTML is static in nature.	XML is dynamic in nature.
HTML is not Case sensitive.	XML is Case sensitive.
HTML tags are predefined tags.	XML tags are user-defined tags.
HTML is used to display the data.	XML is used to store data.
It has an extension of .html and .htm	It has an extension of .xml

## What is XML Used For?

XML is one of the most widely used formats for sharing structured information today

- Between programs
- Between people
- Between computers and people
- Both locally and across networks

## ★ The main components of an XML document are,

1. Elements
2. Content
3. Attributes
4. Comments

## ★ XML Syntax,

### 1. The XML Prolog

The XML document can optionally have an XML declaration. It is called the XML prolog.

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

- The XML declaration is case sensitive and must begin with "<?xml>" where "xml" is written in lowercase.
- If a document contains XML declaration, then it strictly needs to be the first statement of the XML document.
- The XML declaration strictly needs to be the first statement in the XML document.
- An HTTP protocol can override the value of encoding that you put in the XML declaration.

### 2. Tags and Elements

An XML file is structured by several XML elements, also called XML nodes or XML tags. The names of XML elements are enclosed in triangular brackets < > as shown below

```
<element>....</element>
```

### 3. Root Element

An XML document can have only one root element.

```
<root>  
  <x>...</x>  
  <y>...</y>  
</root>
```

### 4. All XML Elements Must Have a Closing Tag

In XML, it is illegal to omit the closing tag. All elements must have a closing tag except prolog. XML prolog does not have a closing tag! This is not an error. The prolog is not a part of the XML document.

```
<message>This is correct</message>
```

### 5. XML Tags are Case Sensitive

XML tags are case sensitive. The tag <Letter> is different from the tag <letter>.

### 6. XML Elements Must be Properly Nested

"Properly nested" simply means that since the <i> element is opened inside the <b> element, it must be closed inside the <b> element.

```
<b><i>This text is bold and italic</i></b>
```

### 7. XML Attribute Values Must Always be Quoted

XML elements can have attributes in name/value pairs just like in HTML. In XML, the attribute values must always be quoted



```

<note date="12/11/2007">
    <to>Tove</to>
    <from>Jani</from>
</note>

```

## 8. Entity References

Some characters have a special meaning in XML. If you place a character like "<" inside an XML element, it will generate an error because the parser interprets it as the start of a new element.

To avoid this error, replace the "<" character with an entity reference

## 9. XML Attributes vs Elements

A given attribute can occur only once within a tag;

Its value is always a string On the other hand tags defining elements/sub elements can repeat any number of times and their values may be string data or sub elements Same data may be encoded using attributes or elements or a combination of the two

```

<person name="Alan" age="42">
    <email>agb@abc.com</email>
</person>

```

```

<person name="Alan">
    <age>42</age>
    <email>agb@abc.com</email>
</person>

```

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

## 10. Comments in XML

The syntax for writing comments in XML is similar to that of HTML.

```
<!--This is a comment-->
```

## 11. XML Naming Rules

XML elements must follow these naming rules:

1. Element names are case sensitive
2. Element names must start with a letter or underscore
3. Element names cannot start with the letters xml (or XML, or Xml, etc)
4. Element names can contain letters, digits, hyphens, underscores, and periods
5. Element names cannot contain spaces

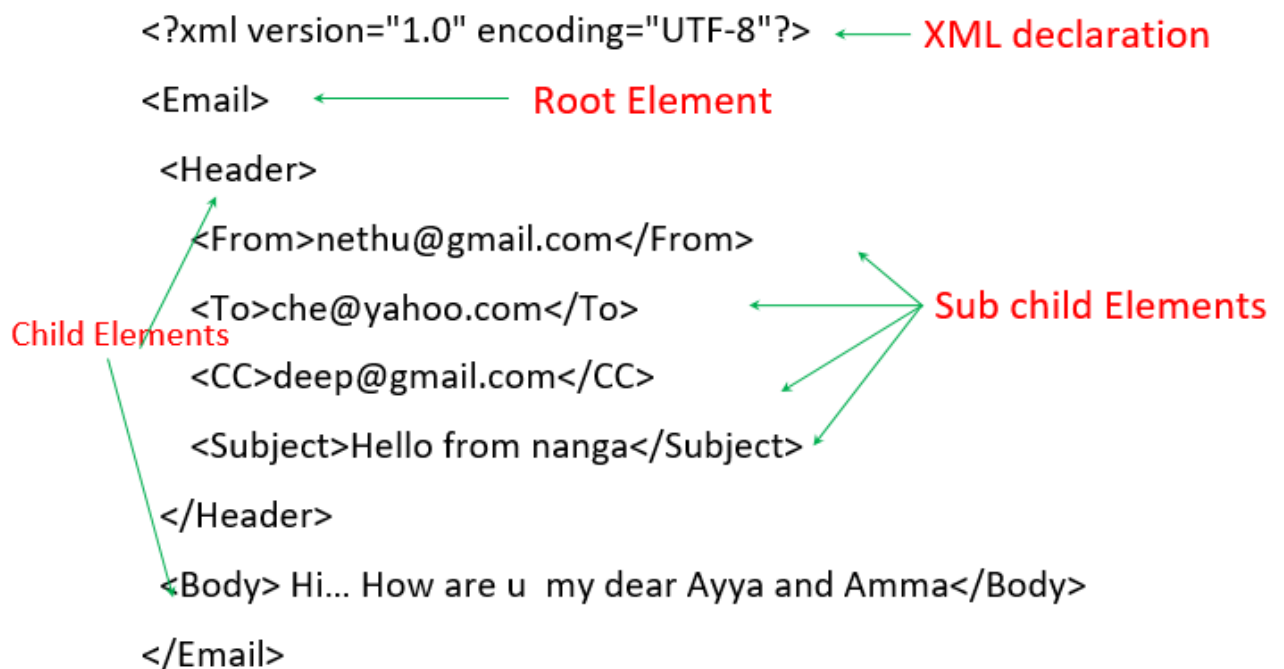
### ★ XML Document Example

```
<?xml version = "1.0"?>
<contact-info>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</contact-info>
```

```
<?xml version="1.0"?>
<contact-info>
  <name>Tanmay Patil</name>
  <company>TutorialsPoint</company>
  <phone>(011) 123-4567</phone>
</contact-info>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<Email>
  <Header>
    <From>nethu@gmail.com</From>
    <To>che@yahoo.com</To>
    <CC>deep@gmail.com</CC>
    <Subject>Hello from nanga</Subject>
  </Header>
  <Body> Hi... How are u my dear Ayya and Amma</Body>
</Email>
```



```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
```

```
<book category="web">
  <title lang="en">Learning XML</title>
  <author>Erik T. Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
</bookstore>
```

## ★ Advantages of Using XML in Enterprise Applications

- Data Interchange and Interoperability
- Platform Independence
- Extensibility and Flexibility
- Structured Data Representation

## ★ XML Processing Technologies

- Document Object Model (DOM)

DOM is an in memory representation of an XML document as a tree structure.

It allows programs to dynamically access, modify, and manipulate XML data.

Example: In Java, DOM parsers like JAXP provide APIs to traverse and manipulate XML documents using DOM.

- Simple API for XML (SAX)
- XML Query Languages (XPath, XQuery)
- XML Transformation Technologies (XSLT)

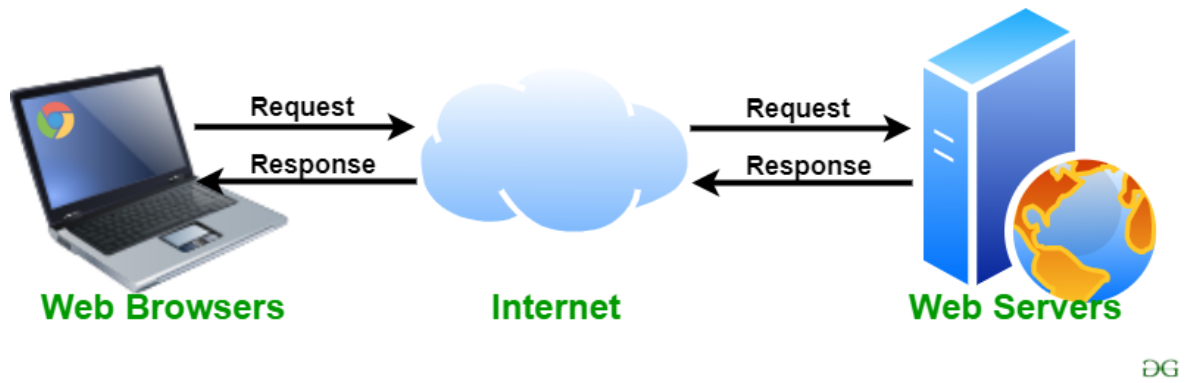
## ★ Web Servers

Web servers are primarily used to process and manage HTTP/HTTPS requests and responses from the client system.

Web servers are software programs responsible for serving static content to web browsers over the HTTP protocol.

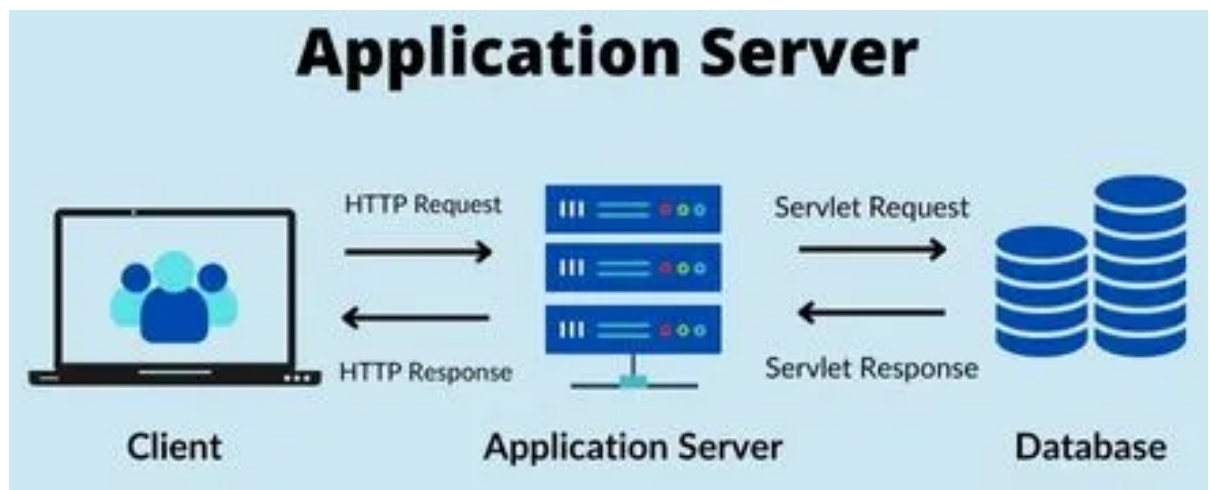
Functionality: They handle requests from clients (browsers) and respond with static HTML, CSS, JavaScript files, images, etc.

Examples: Apache HTTP Server, Nginx, Microsoft IIS



### ★ Application Servers

Application servers are software frameworks or platforms designed to execute dynamic content and business logic for web applications.



They support the execution of server side code (e.g., Java, PHP, .NET) and provide services such as database access, transaction management, and session handling.

Examples: Tomcat, JBoss ( WildFly ), IBM WebSphere, Oracle WebLogic.

### ★ Integrated Development Environment (IDE)

An Integrated Development Environment (IDE) is a software suite that provides essential tools for writing, testing, and debugging code.

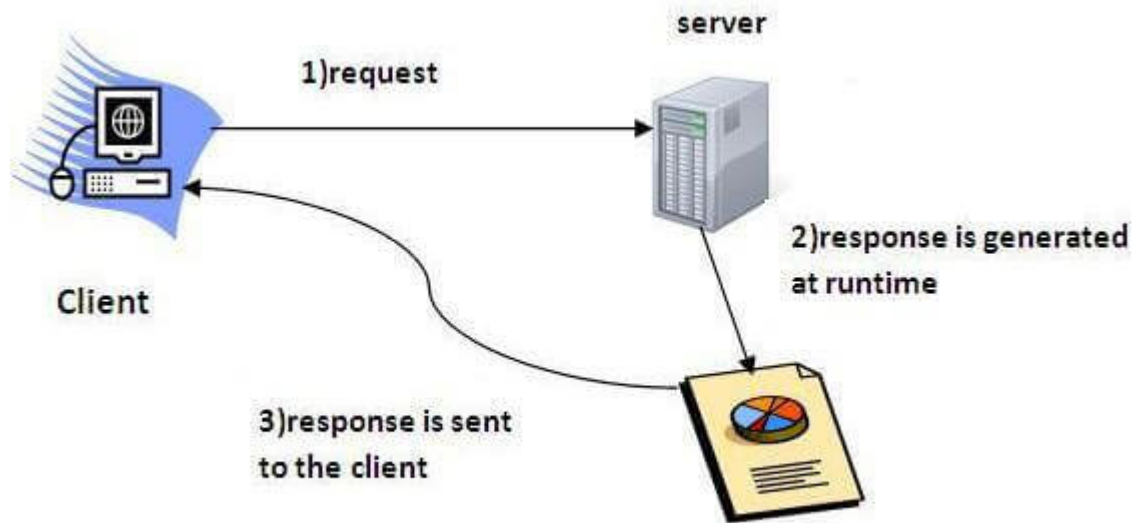
- Visual Studio Code
- NetBeans
- IntelliJ IDEA
- Eclipse

### ★ What is a Servlet ?

Servlet is a technology which is used to create a web application.

Servlet is an API that provides many interfaces and classes including documentation.

Servlet is a web component that is deployed on the server to create a dynamic web page.



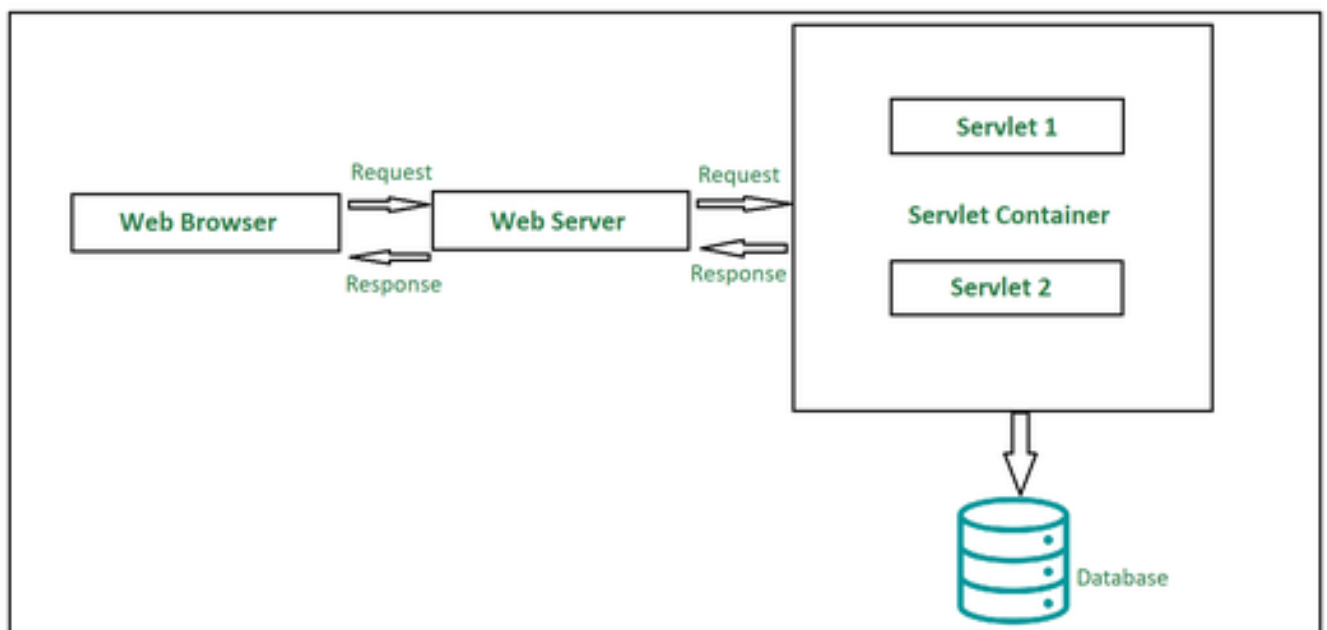
### ★ What is a web application ?

A web application is an application accessible from the web.

A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript.

### ★ Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.



### ★ The Java Servlet Life cycle

includes three stages right from its start to the end until the Garbage Collector clears it. These three stages are described below.

#### 1. Initialization Phase

The servlet is initialized by calling the `init()` method.

#### 2. Service Phase

The servlet calls `service()` method to process a client's request.

#### 3. Destruction Phase

The servlet is terminated by calling the `destroy()` method.

Following is the sample source code structure of a servlet example to show Hello World

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

    private String message;

    public void init() throws ServletException {
        // Do required initialization
        message = "Hello World";
    }

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

        // Set response content type
        response.setContentType("text/html");

        // Actual logic goes here.
        PrintWriter out = response.getWriter();
        out.println("<h1>" + message + "</h1>");
    }

    public void destroy() {
        // do nothing.
    }
}
```

## ★ Java Servlet example

### GET and POST Requests

In the index.html file, create two HTML forms.

**GET:** Data visible in URL, size limitations, suitable for non-sensitive data.

**POST:** Data sent in request body, not visible in URL, suitable for sensitive data.

**<fieldset>** and **<legend>** Again, these tags group related elements in a form and provide a caption for the group, indicating this form uses the POST method.

```

<div>GET AND POST METHODS</div>

<fieldset>
  <legend>GET</legend>
  <form name="login" action="login" method="GET">
    <label>First Name</label>
    <input type="text" name="fname"/>
    <br><br>
    <label>Last Name</label>
    <input type="text" name="lname"/>
    <br><br>
    <input type="submit" value="Submit"/>
  </form>
</fieldset>
<br><br>
<fieldset>
  <legend>POST</legend>
  <form name="login" action="login" method="post">
    <label>UserName</label>
    <input type="text" name="uname"/>
    <br><br>
    <label>Password</label>
    <input type="password" name="pass"/>
    <br><br>
    <input type="submit" value="Login"/>
  </form>
</fieldset>

```

Create a "login" Servlet Page. The Servlet Page name needs to be added to the appropriate HTML form's action attribute.

Update **doGet** and **doPost** Methods in Created Servlet Page File

→ **The doGet() Method**

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by the doGet() method.

→ **The doPost() Method**

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.



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

    String fname = request.getParameter("fname");
    String lname = request.getParameter("lname");
    String full_Name = fname + " " + lname;

    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code. */
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet login</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h4>First Name " + fname + "</h4>");
        out.println("<h4>Last Name " + lname + "</h4>");
        out.println("<h4>Full Name " + full_Name + "</h4>");
        out.println("</body>");
        out.println("</html>");
    }
}
```



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

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

    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code. */
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet login</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h4>UserName " + uname + "</h4>");
        out.println("<h4>Password " + pass + "</h4>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

**getParameter()** – You call the **request.getParameter()** method to get the value of a form parameter.

Run your Project and Click the Submit Button.

#### GET AND POST METHODS

GET

First Name

Last Name

View the result

← ↻ ⓘ localhost:8080/Web\_APP/login?fname=Nadeera&lname=Bandaranayake

📁 Import favorites | 📁 My Web | 📁 The NeoSmart Files | 🌐 Google

**First Name Nadeera**

**Last Name Bandaranayake**

**Full Name Nadeera Bandaranayake**

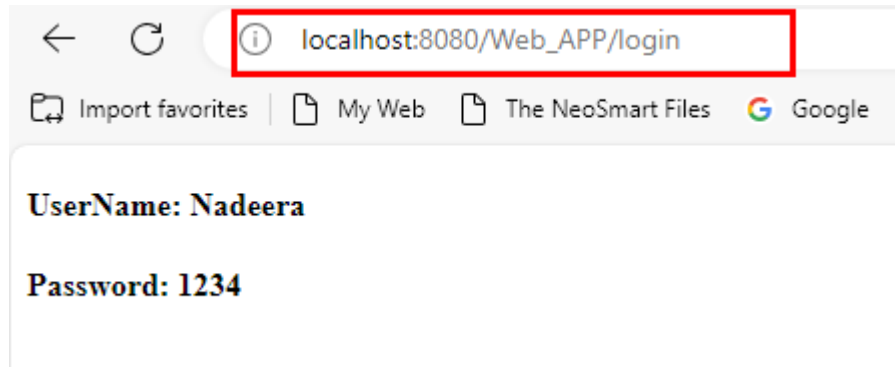
Run your Project and Click the Login Button.

POST

UserName

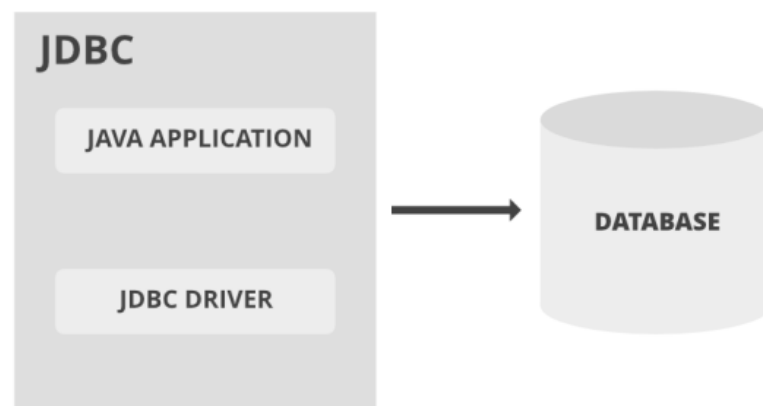
Password

View the result



### ★ What is JDBC (Java Database Connectivity) ?

JDBC is an API(Application programming interface) used in Java programming to interact with databases.



We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

### ★ Servlet with JDBC (Java Database Connectivity)

Create a Login Page. Ex. login.html

```

<html>
  <head>
    <title>JAVA Servlet</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <form name="login" action="login" method="post">
      <label>UserName</label>
      <input type="text" name="uname"/>
      <br><br>
      <label>Password</label>
      <input type="password" name="pass"/>
      <br><br>
      <input type="submit" value="Login"/>
    </form>
  </body>
</html>

```

### Create a MySQL Database and Table

Database Name: **java\_test**

Table Name: **users**

```

CREATE TABLE `users` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL,
  `email` varchar(30) NOT NULL,
  `password` varchar(20) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 COLLATE=utf8_general_ci;

```

### Create a “DBConnection” Java class to connect MySQL databases.

The dbCon method is responsible for establishing a connection with the MySQL database and returning the connection.

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DBConnection {
    private static Connection con;

    public static Connection dbCon(){

        try{
            final String connectionUrl = "jdbc:mysql://localhost:3306/java_test?serverTimezone=UTC";
            final String user="nadeera";
            final String password="Admin@1234";
            con = DriverManager.getConnection(connectionUrl, user, password);
        }
        catch(SQLException ex){
            System.out.println("Error"+ex);
        }

        return con;
    }
}

```

### Create a “Validate” Java class to validate users.

The LoginValidate function obtains the password and username as parameters, then verifies them in the Users table.

The method, Returns true if the username and password are in the table; returns false otherwise.

```
import java.sql.*;

public class Validate {
    public static boolean loginValidate(String username, String pass) {

        boolean valid= false;

        try{

            //Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DBConnection.dbCon();
            Statement stmt = conn.createStatement();
            String query="SELECT * FROM users WHERE name='"+username+"' AND password='"+pass+"'";
            ResultSet rs = stmt.executeQuery(query);

            valid = rs.next();
        }
        catch(SQLException e){//| ClassNotFoundException
            System.out.println("Error: "+e);
        }

        return valid;
    }
}
```

### **“Login” Servlet Page**

```

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

    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();

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

    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet login</title>");
    out.println("</head>");
    out.println("<body>");

    if (Validate.loginValidate(uname, pass)) {
        response.sendRedirect("welcome");
    } else {
        out.print("Login Failed");
    }
    out.println("</body>");
    out.println("</html>");

}

```

The doPost method get the username and password from the login.html form and sends them to the login validate method.

If the login validate method returns true this will redirect to the welcome page otherwise it print Login Failed.

## ★ Sessions and Cookies in Servlet

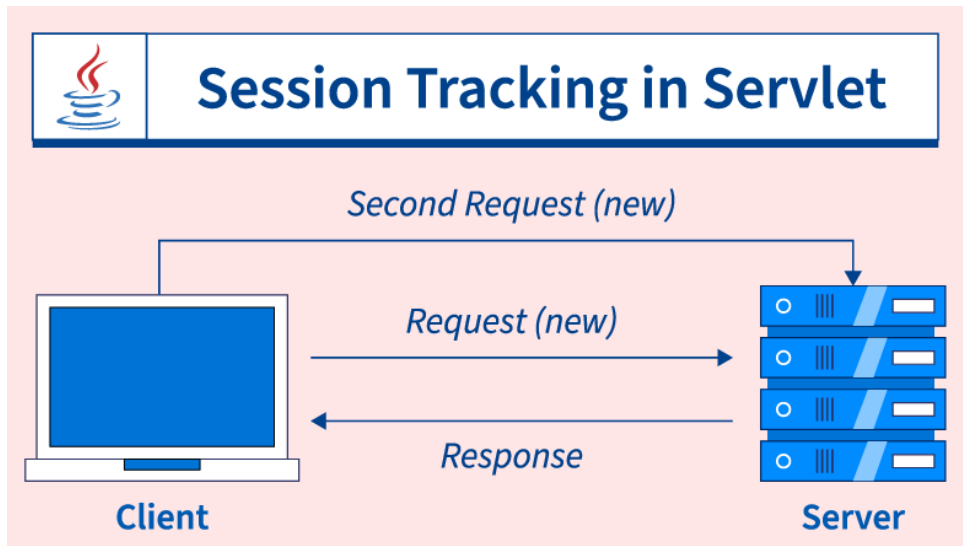
### Session

A session is a period of logical connection or interaction between a web client (like a browser) and a web server.

Session is used to save user information on the server.

Sessions are used to store user-specific data across multiple requests, allowing web applications to maintain stateful interactions with clients.

Sessions can be used for authentication and authorization purposes.



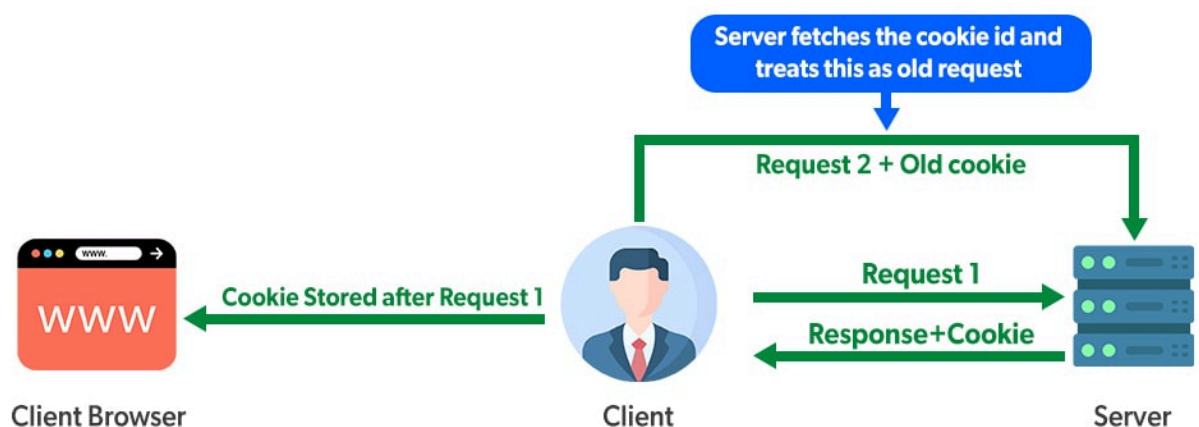
## Cookie

Cookies are the textual information that is stored in key-value pair format to the client's browser during multiple requests.

A cookie is a small piece of information that is persisted between the multiple client requests.

Cookies are commonly used to store user preferences, session identifiers, authentication tokens, and other client-specific information.

Cookies have limitations such as size restrictions and security concerns, so sensitive information should not be stored directly in cookies.



## ★ How to using Cookies and Sessions

### Update the Login Page with Cookies and Sessions

```

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

    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();

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

    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet login</title>");
    out.println("</head>");
    out.println("<body>");

    if (Validate.loginValidate(uname, pass)) {

        //Create Session
        HttpSession session = request.getSession();
        session.setAttribute("username", uname);

        //Setting a Cookie
        Cookie ck = new Cookie("cname", uname);
        ck.setMaxAge(60 * 60);
        response.addCookie(ck);

        response.sendRedirect("welcome");
    } else {
        out.print("Login Failed");
    }

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

}

```

- Create a session object if it is already not created.  
**HttpSession session = request.getSession();**
- store the user information into the session object - setAttribute(String name, Object value)  
**session.setAttribute("username", uname);**
- Creating a Cookie object - Cookie cookie = new Cookie("key","value");  
**Cookie ck = new Cookie("cname", uname);**
- Setting the maximum age – You use setMaxAge to specify how long (in seconds) the cookie should be valid.  
**ck.setMaxAge(60 \* 60);**
- Sending the Cookie into the HTTP response headers  
**response.addCookie(ck);**



## Update the Welcome Page

Process requests method used for both HTTP GET and POST methods.

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

    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();

    out.println("<!DOCTYPE html>");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet welcome</title>");
    out.println("</head>");

    HttpSession session = request.getSession(false);
    //Check the Session
    if (session == null || session.getAttribute("username") == null) {
        response.sendRedirect("login.html");
    } else {

        //get cname from cookie
        Cookie cookies[] = request.getCookies();
        if (cookies != null) {
            String name = cookies[0].getValue();
            for (int i = 0; i < cookies.length; i++) {
                if ("cname".equals(cookies[i].getName())) {
                    name = cookies[i].getValue();
                }
            }
            out.println("<h4>Hi," + name + "</h4>");
        }
        out.println("<body>");
        out.println("<h1>Dashboard</h1>");
        out.println("<a href='Logout'>Log out</a>");
    }
    out.println("</body>");
    out.println("</html>");
}
```

The page loads the current HttpSession associated with this request, and if there is no current session or the session username is null, it will redirect to the login page.

Also, it retrieves the value of the "cname" cookie from the cookies and prints its value on the webpage.

## Create a Logout Page

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

    response.setContentType("text/html");
    PrintWriter out=response.getWriter();

    request.getRequestDispatcher("login.html").include(request, response);

    HttpSession session=request.getSession();
    session.invalidate();

    out.print("You are successfully logged out!");

    out.close();
}
```

Once the user requests to logout, we need to destroy that session. To do this, we need to use `invalidate()` method in the `HttpSession` Interface.

After the session is destroyed, the page will redirect to the `login.html` page.

**RequestDispatcher** is an interface provided by the Servlet API. It allows servlets to forward requests or include responses to other servlets essentially, it helps in dispatching the client's request from one servlet to another or including content from another resource into the response.

### ★ Difference Between Session and Cookies

Cookie	Session
Cookies are client-side files on a local computer that hold user information.	Sessions are server-side files that contain user data.
Cookies end on the lifetime set by the user.	When the user quits the browser or logs out of the program, the session is over.
It can only store a certain amount of info.	It can hold an indefinite quantity of data.
Cookies are not secured.	Sessions are more secure compared to cookies.
Cookies stored data in text files.	Session save data in encrypted form.
Cookies stored on limited data.	Session stored unlimited data.

### ★ Servlet API Overview

The Servlet API provides classes and interfaces for developing Servlets. Servlet API has 2 packages namely,

1. `javax.servlet`
2. `javax.servlet.http`

## ★ Advantages of Using Servlets

1. **Platform Independence:** Servlets run on any server supporting Java.
2. **Robustness:** Built in exception handling and error management.
3. **Scalability:** Handle multiple requests concurrently.
4. **Session Management:** Maintain stateful communication with clients.
5. **Database Connectivity:** Seamlessly integrate with databases via JDBC.
6. **Security:** Implement authentication, authorization, and data encryption.

## ★ What is Session management ?

Session management refers to the process of controlling and maintaining user sessions in web applications.

## ★ What is State Management ?

State management in web applications refers to the process of preserving and managing data and application state across multiple user interactions.

It involves storing and retrieving user specific information, session data, and application state variables.

### Types of State Management:

#### 1. Client Side State Management

- Data stored on the client side (browser) using cookies, local storage, or session storage.
- Suitable for small amounts of data and maintaining user preferences.

#### 2. Server Side State Management

- Data stored on the server side, typically in session objects or databases.
- Used for managing user sessions, application state, and complex data interactions.

## ★ Advantages of Client Side State Management

- **Reduced server load:** Offloads data storage and management tasks to the client side.
- **Faster access:** Client side data retrieval is often faster than server side requests.
- **Improved user experience:** Enables personalized interactions and offline capabilities.

## ★ Advantages of Server Side State Management

- **Enhanced security:** Data stored on the server is more secure than client side storage.
- **Scalability:** Server side solutions can handle large amounts of data and concurrent users.
- **Centralized management:** Simplifies data access, retrieval, and manipulation on the server.

## ★ Service-Oriented Architecture (SOA)

Service-Oriented Architecture (SOA) is a stage in the evolution of application development and/or integration.

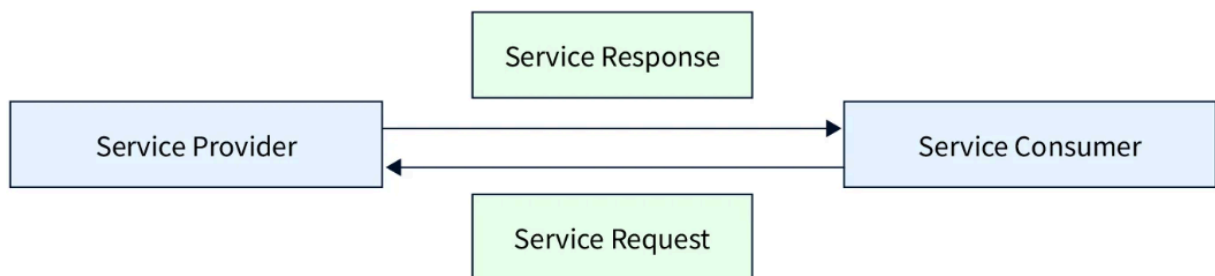
Service-oriented architecture (SOA) is a method of software development that uses software components called services to create business applications.

It defines a way to make software components reusable using the interfaces.

Some SOA product has been built by Oracle (SOA Suite), IBM(Websphere), Microsoft(BizTalk)

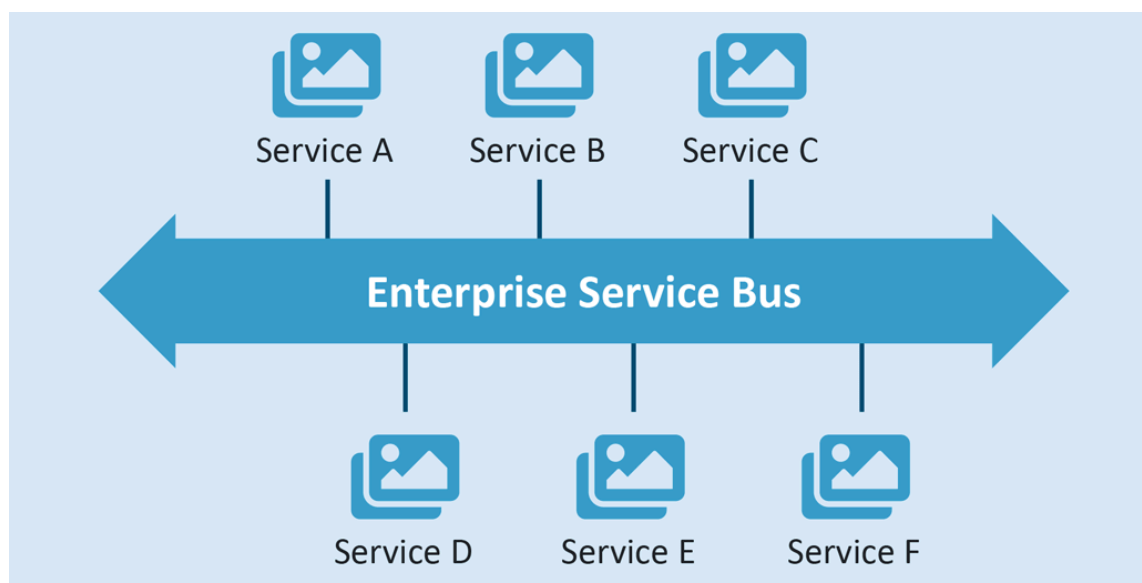
### ★ What are the two major roles within SOA?

1. Service consumer
2. Service provider



## ESB (Enterprise Service Bus)

The idea of an ESB is that many components can connect in a standardized way and then communicate over the bus with any other component.



## ★ Advantages of SOA

- **Service reusability:** In SOA, applications are made from existing services. Thus, services can be reused to make many applications.
- **Easy maintenance:** As services are independent of each other they can be updated and modified easily without affecting other services.
- **Platform independent:** SOA allows making a complex application by combining services picked from different sources, independent of the platform.
- **Availability:** SOA facilities are easily available to anyone on request.
- **Reliability:** SOA applications are more reliable because it is easy to debug small services rather than huge codes
- **Scalability:** Services can run on different servers within an environment, this increases scalability

## ★ Disadvantages of SOA

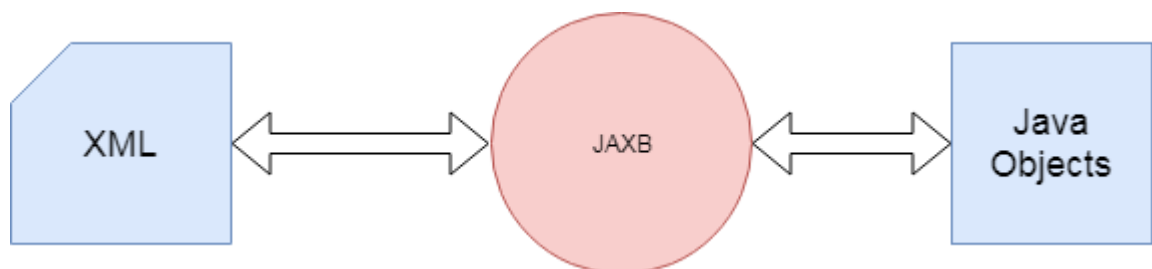
- **High investment:** A huge initial investment is required for SOA.
- **Complex service management:** When services interact they exchange messages to tasks. the number of messages may go in millions. It becomes a cumbersome task to handle a large number of messages.
- **Increase the Network Traffic**

## ★ JAXB (Java Architecture for XML Binding)

JAXB stands for Java Architecture for XML Binding.

It provides a mechanism to marshal (write) java objects into XML and unmarshal (read) XML into objects.

It is used to convert java objects into xml.



## ★ Java Annotation

Java Annotation is a tag that represents the metadata. i.e. attached with class, interface, methods or fields to indicate some additional

**@XmlRootElement** : The name of the root XML element is derived from the class name, and we can also specify the name of the root element of the XML using its name attribute.

**@XmlType** : define the order in which the fields are written in the XML file

**@XmlElement** : define the actual XML element name that will be used

**@XmlAttribute** : define the id field is mapped as an attribute instead of an element

**@XmlTransient** : annotate fields that we don't want to be included in XML

## ★ Read and Write XML File Using JAXB

Create a new class named "Student."

```
import javax.xml.bind.annotation.*;

@XmlType(propOrder = { "name", "batch" })

public class Student {
    private String regno;
    private String name;
    private int batch;
    private String ati;

    public String getRegno() {
        return regno;
    }

    @XmlAttribute
    public void setRegno(String regno) {
        this.regno = regno;
    }

    public String getName() {
        return name;
    }

    @XmlElement(name = "name")
    public void setName(String name) {
        this.name = name;
    }

    public int getBatch() {
        return batch;
    }

    @XmlElement(name = "batch")
    public void setBatch(int batch) {
        this.batch = batch;
    }

    public String getAti() {
        return ati;
    }

    @XmlTransient
    public void setAti(String ati) {
        this.ati = ati;
    }
}
```

**Create another new class named " SLIATE."**

```
import java.util.ArrayList;
import javax.xml.bind.annotation.*;

@XmlRootElement(name = "sliate")
@XmlAccessorType(XmlAccessType.FIELD)

public class SLIATE {

    @XmlElement(name = "student")
    private ArrayList<Student> studentList;

    public ArrayList<Student> getStudentList() {
        return studentList;
    }

    public void setStudentList(ArrayList<Student> studentList) {
        this.studentList = studentList;
    }
}
```

**Import following packages to your main Class**

```
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
```

## Create a Method for Write a XML File (JAXB Marshalling Example: Converting Object into XML)

```
public static void marshal(){
    try{

        SLIATE kandyATI = new SLIATE();
        ArrayList<Student> lst=new ArrayList<>();

        Student stu1 = new Student();
        stu1.setRegno("PT001");
        stu1.setName("Nadeera");
        stu1.setBatch(2023);
        stu1.setAti("Kandy");
        lst.add(stu1);

        Student stu2 = new Student();
        stu2.setRegno("PT002");
        stu2.setName("Nuwan");
        stu2.setBatch(2023);
        stu2.setAti("Kandy");
        lst.add(stu2);

        kandyATI.setStudentList(lst);

        JAXBContext context = JAXBContext.newInstance(SLIATE.class);
        Marshaller mar= context.createMarshaller();
        mar.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
        mar.marshal(kandyATI, new File("./ati_students.xml"));
    }
    catch(JAXBException ex){
        System.out.println("Error!"+ex);
    }
}
```

## Create a Method for Read the XML File JAXB UnMarshalling Example: Converting XML into Object



```

public static void unmarshal() {
    try {
        JAXBContext context = JAXBContext.newInstance(SLIATE.class);
        Unmarshaller um = context.createUnmarshaller();

        SLIATE sliate = (SLIATE) um.unmarshal(new FileReader("./ati_students.xml"));
        ArrayList<Student> list = sliate.getStudentList();

        for (Student stu : list) {
            System.out.println("Name: " + stu.getName() + " Batch "
                               + stu.getBatch());
        }

    } catch (JAXBException | IOException ex) {
        System.out.println("Error!" + ex);
    }
}

```

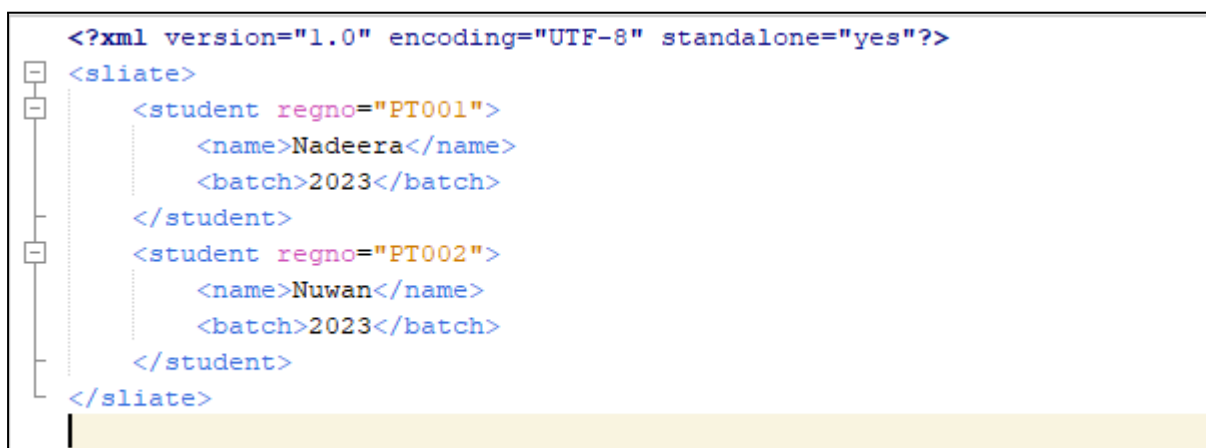
### Invoke Write and Read Methods

```

public static void main(String[] args) {
    marshal();
    unmarshal();
}

```

### Output - 1

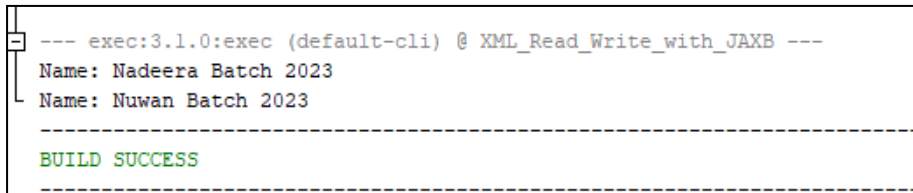


```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<sliate>
  <student regno="PT001">
    <name>Nadeera</name>
    <batch>2023</batch>
  </student>
  <student regno="PT002">
    <name>Nuwan</name>
    <batch>2023</batch>
  </student>
</sliate>

```

### Output - 2



```

--- exec:3.1.0:exec (default-cli) @ XML_Read_Write_with_JAXB ---
Name: Nadeera Batch 2023
Name: Nuwan Batch 2023
-----
BUILD SUCCESS
-----

```

## ★ JDBC Extra code example

### Create a Method for Insert Data to the MySQL table

```
static void addData() {
    try {
        Connection con = DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);
        Statement stmt = con.createStatement();
        String query = "INSERT INTO users (name,email) VALUES('Nadeera','nadeera@gmail.com')";
        stmt.executeUpdate(query);
        System.out.println("User added successfull..!");

        con.close();

    } catch (SQLException e) {
        System.out.println("Error: " + e);
    }
}
```

### Create a Method for Select Data to from MySQL table

```
static void getData(){
    try{
        Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);
        Statement stmt = conn.createStatement();
        String query = "SELECT * FROM users";
        ResultSet rs = stmt.executeQuery(query);

        while(rs.next()){
            System.out.println("ID : "+rs.getInt("id"));
            System.out.println("Name : "+rs.getString("name"));
            System.out.println("Email : "+rs.getString("email"));
        }

        conn.close();
    }
    catch (SQLException e) {
        System.out.println("Error: " + e);
    }
}
```

### Create a Method for Update Data

```
static void updateUser(){
    try{
        Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);
        Statement stmt = conn.createStatement();

        String query = "UPDATE users SET email='abc@gmail.com' WHERE name='Nadeera'";
        stmt.executeUpdate(query);
        System.out.println("User update successfull..!");

        stmt.close();
        conn.close();
    }
    catch (SQLException e) {
        System.out.println("Error: " + e);
    }
}
```

### Create a Method for Delete Data

```
static void deleteUser(){
    try{
        Connection conn = DriverManager.getConnection(DB_URL, DB_USER, DB_PASS);
        Statement stmt = conn.createStatement();

        String query = "DELETE FROM users WHERE id=2";
        stmt.executeUpdate(query);
        System.out.println("User Delete successfull..!");

        stmt.close();
        conn.close();
    }
    catch (SQLException e) {
        System.out.println("Error: " + e);
    }
}
```

## ★ What is DTD (Document Type Definition) ?

A DTD defines the structure and the legal elements and attributes of an XML document.

An application can use a DTD to verify that XML data is valid.

### XML document with an internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

The DTD above is interpreted like this:

- **!DOCTYPE note** defines that the root element of this document is note
- **!ELEMENT note** defines that the note element must contain four elements: "to,from,heading,body"
- **!ELEMENT to** defines the to element to be of type "#PCDATA"
- **!ELEMENT from** defines the from element to be of type "#PCDATA"
- **!ELEMENT heading** defines the heading element to be of type "#PCDATA"
- **!ELEMENT body** defines the body element to be of type "#PCDATA"

Tip: #PCDATA means parseable character data.

### XML document with a reference to an external DTD

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

And here is the file "note.dtd", which contains the DTD:

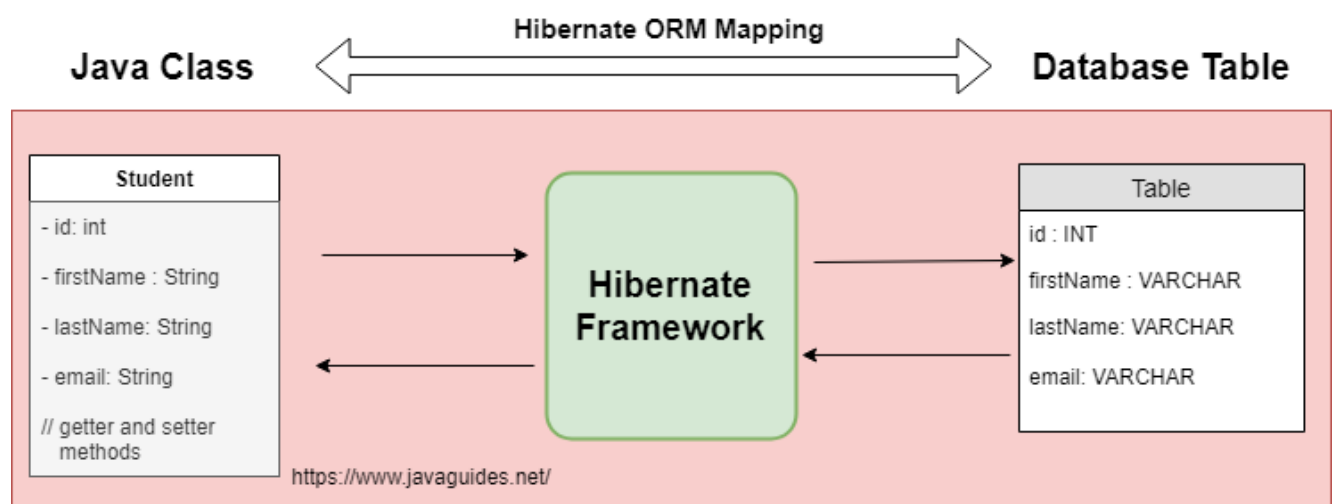
```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

## ★ What is Hibernate ?

Hibernate is a Java framework that simplifies the development of Java applications to interact with the database.

Hibernate is a popular open-source Object-Relational Mapping (ORM) framework for Java applications.

It simplifies the development of database-driven applications by providing a mapping between Java objects and database tables, allowing developers to work with objects rather than SQL queries directly.



## Object-Relational Mapping (ORM)

ORM is a programming technique that maps object-oriented models to relational database tables.

Hibernate automates the mapping process, eliminating the need for developers to write manual SQL queries for database operations.

## Advantages of Hibernate Framework

1. Open Source and Lightweight
2. Fast Performance
3. Automatic Table Creation
4. Database Independent Query
5. Simplifies Complex Join
6. Provides Query Statistics and Database Status

### explain a given hibernated class code.

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "students")
public class Student {
    @Id
    private int id;
    private String name;
    private int age;

    // Getters and Setters
    public int getId() {
        return id;
    }

    public void setId (int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

### Explanation:

- **@Entity**: Marks this class as a Hibernate entity.
- **@Table (name = "students")**: Specifies the database table name this entity maps to.
- **@Id**: Indicates that the id field is the primary key.
- **Fields (id, name, age)**: These fields will be mapped to the corresponding columns in the database table.
- **Getters and Setters**: Methods to access and modify the fields.

### ★ Write a few getters and setters for a class ?

```
public class Person {  
    private String name;  
    private int age;  
  
    // Getter for name  
    public String getName() {  
        return name;  
    }  
  
    // Setter for name  
    public void setName(String name) {  
        this.name = name;  
    }  
  
    // Getter for age  
    public int getAge() {  
        return age;  
    }  
  
    // Setter for age  
    public void setAge(int age) {  
        this.age = age;  
    }  
}
```

### ★ Write a method for a requirement.

```
public class Person {  
    private String name;  
    private int age;  
  
    // Existing getters and setters...  
    // Method to calculate the year of birth  
    public int calculateYearOfBirth() {  
        int currentYear = java.time.Year.now().getValue();  
        return currentYear - age;  
    }  
}
```

### ★ Enterprise Java Beans (EJB)

Enterprise Java Beans (EJB) is one of the several Java APIs for standard manufacture of enterprise software.

EJB is a server-side software element that summarizes the business logic of an application.

#### Disadvantages of EJB

1. Requires application server
2. Requires only Java clients. For other language clients, you need to go for web service.
3. Complex to understand and develop EJB applications.

## ★ Session Bean

Session bean encapsulates business logic only, it can be invoked by local, remote and web service clients.

It can be used for calculations, database access etc.

The life cycle of session bean is maintained by the application server (EJB Container).

### Types of Session Bean

1. **Stateless Session Bean:** It doesn't maintain the state of a client between multiple method calls.
2. **Stateful Session Bean:** It maintains state of a client across multiple requests.
3. **Singleton Session Bean:** One instance per application, it is shared between clients and supports concurrent access.

## ★ Difference between Servlet and Enterprise Java Bean (EJB)

Servlet	Enterprise Java Bean (EJB)
Handling web-based applications	Managing complex business logic and transactions
Managed by servlet container (e.g., Tomcat)	Managed by EJB container
Java classes that handle HTTP requests/responses	Server-side components encapsulating business logic
No built-in management, needs manual handling	Built-in support for declarative and programmatic management
Using Basic security features (RBAC)	Using Advanced security features, including JAAS
Lifecycle Methods - init(), service(), destroy()	@PostConstruct, @PreDestroy, various callback methods