**WT Endsem 2025 - Q & A (PYQ – 2024, 2023, 2022)**

**UNIT 3**

**Q1. Explain doGet() & dopost() methods of servet. Differentiate do Get Vs do Post (Min 04). [9]**

**=>**

**✅ doGet() and doPost() in Servlets**

In **Java Servlets**, doGet() and doPost() are methods of the **HttpServlet** class, used to handle **HTTP GET and POST requests** respectively.

**📘 1. doGet() Method:**

* The doGet() method is used to **handle GET requests** from a client.
* GET requests typically **send data via URL** parameters.
* Mostly used for **retrieving data**, not modifying server resources.

**🧾 Syntax:**

java

CopyEdit

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Processing code

}

**✅ Example:**

java

CopyEdit

String name = request.getParameter("username");

response.getWriter().println("Welcome " + name);

**📗 2. doPost() Method:**

* The doPost() method handles **POST requests**, where data is sent in the **request body**.
* It is used for **sending sensitive data**, form submissions, or uploading files.
* Data is **not visible in the URL**.

**🧾 Syntax:**

java

CopyEdit

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

// Processing code

}

**✅ Example:**

java

CopyEdit

String email = request.getParameter("email");

response.getWriter().println("Email registered: " + email);

**📊 3. Difference Between doGet() and doPost():**

| **Parameter** | **doGet()** | **doPost()** |
| --- | --- | --- |
| **1. Data Transmission** | Sent via URL (query string) | Sent in request body (invisible) |
| **2. Security** | Less secure (data visible in URL) | More secure (data not shown in URL) |
| **3. Size Limitation** | Limited (~2 KB depending on browser) | No significant size limit |
| **4. Use Case** | Retrieving data (e.g., search query) | Submitting data (e.g., registration) |
| **5. Caching** | Can be cached by browser | Not cached |
| **6. Bookmarking** | Can be bookmarked | Cannot be bookmarked |

**✅ 4. Servlet Class Using Both doGet() and doPost():**

java

CopyEdit

public class SampleServlet extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.getWriter().println("GET method called");

}

protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.getWriter().println("POST method called");

}

}

**Q2. What is XML DTDs? Explain with example. Differentiate XML DTDs Vs XML schema (Min.04). [9]**

**=>**

## ✅ XML DTD (Document Type Definition)

### 📘 Definition:

**DTD** stands for **Document Type Definition**.  
It defines the **structure** and the **legal elements** and **attributes** of an XML document.

* A DTD can be **internal** (defined within XML) or **external** (referenced via URL or file).
* It acts as a **rulebook** for validating XML content to ensure it follows a specific format.

### 🧾 Syntax of Internal DTD:

xml

CopyEdit

<!DOCTYPE student [

<!ELEMENT student (name, age)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT age (#PCDATA)>

]>

<student>

<name>Rahul</name>

<age>21</age>

</student>

### 🧾 Syntax of External DTD:

**File: student.dtd**

dtd

CopyEdit

<!ELEMENT student (name, age)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT age (#PCDATA)>

**File: student.xml**

xml

CopyEdit

<!DOCTYPE student SYSTEM "student.dtd">

<student>

<name>Rahul</name>

<age>21</age>

</student>

### ✅ Explanation:

* <!ELEMENT> is used to define each element and its contents.
* #PCDATA stands for **Parsed Character Data**, meaning the element contains text.
* If an XML document conforms to its DTD, it is called **valid** XML.

## 📊 Difference: XML DTD vs XML Schema

| **Aspect** | **XML DTD** | **XML Schema** |
| --- | --- | --- |
| **1. Syntax Language** | Written in its own DTD syntax | Written in **XML syntax** |
| **2. Data Types** | Limited (only text – #PCDATA, CDATA) | Rich support (int, float, date, etc.) |
| **3. Namespace Support** | Not supported | Fully supported |
| **4. Extensibility** | Less extensible | Highly extensible |
| **5. Validation Strength** | Basic validation | Strong validation (type, pattern) |
| **6. Industry Use** | Older and simpler | Modern and preferred |

**Q3. Explain the servlet lifecycle. Explain session management using cookies and URL Rewriting. [12]**

**=>**

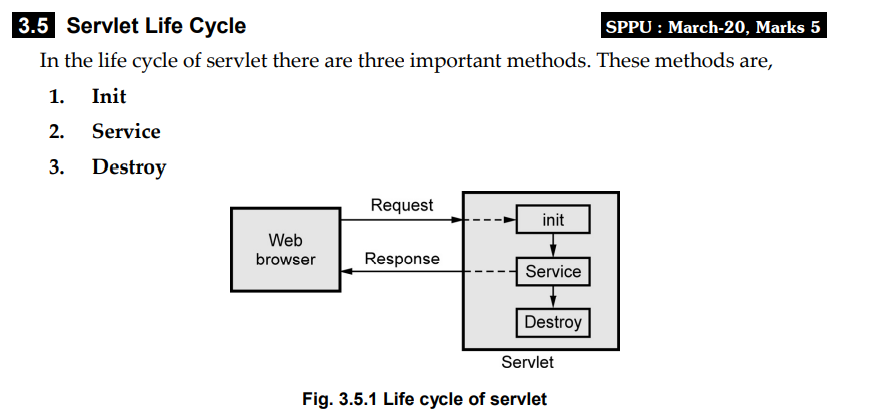
## ✅ Part 1: Servlet Lifecycle

### 📘 Definition:

The **Servlet Lifecycle** defines the **entire process from servlet creation to destruction** on the server.  
It is controlled by the **Servlet Container (like Tomcat)** and includes **3 main methods** defined in javax.servlet.Servlet interface:

### 🔁 Lifecycle Phases:

| **Phase** | **Method** | **Description** |
| --- | --- | --- |
| **1. Initialization** | init() | Called once when the servlet is loaded. Used to initialize resources. |
| **2. Request Handling** | service() | Called every time the servlet receives a request. |
| **3. Destruction** | destroy() | Called once when servlet is unloaded. Used to release resources. |



### 📌 Servlet Lifecycle Flowchart:

plaintext

CopyEdit

Server Starts

↓

Servlet Class Loaded

↓

→ init() method called

↓

→ service() method called on each request

↓

Handles doGet() / doPost()

↓

→ destroy() method called (on shutdown)

### 🧾 Example:

java

CopyEdit

public class MyServlet extends HttpServlet {

public void init() {

System.out.println("Servlet Initialized");

}

public void doGet(HttpServletRequest req, HttpServletResponse res)

throws ServletException, IOException {

res.getWriter().println("Hello from Servlet!");

}

public void destroy() {

System.out.println("Servlet Destroyed");

}

}

## ✅ Part 2: Session Management

Session management is used to **maintain state** (data) across multiple HTTP requests, since HTTP is **stateless**.

### 🍪 A. Using Cookies

#### 📘 What is a Cookie?

* A **cookie** is a small piece of data sent by the server to the client and stored on the client browser.
* Sent back to the server with every subsequent request.

#### 🧾 Example in Servlet:

java

CopyEdit

// Create Cookie

Cookie cookie = new Cookie("username", "Rahul");

response.addCookie(cookie);

// Read Cookie

Cookie[] cookies = request.getCookies();

for (Cookie c : cookies) {

if (c.getName().equals("username")) {

out.println("Welcome " + c.getValue());

}

}

#### ✅ Features:

* Stored on client-side
* Limited size (~4 KB)
* Used for personalization & login tracking

### 🔗 B. Using URL Rewriting

#### 📘 What is URL Rewriting?

* Appends session data **directly to the URL** as **query parameters**.
* Useful when cookies are disabled on the client.

#### 🧾 Example:

java

CopyEdit

// Sending data in URL

response.sendRedirect("dashboard.jsp?username=Rahul");

// Receiving data

String user = request.getParameter("username");

out.println("Welcome " + user);

#### ✅ Features:

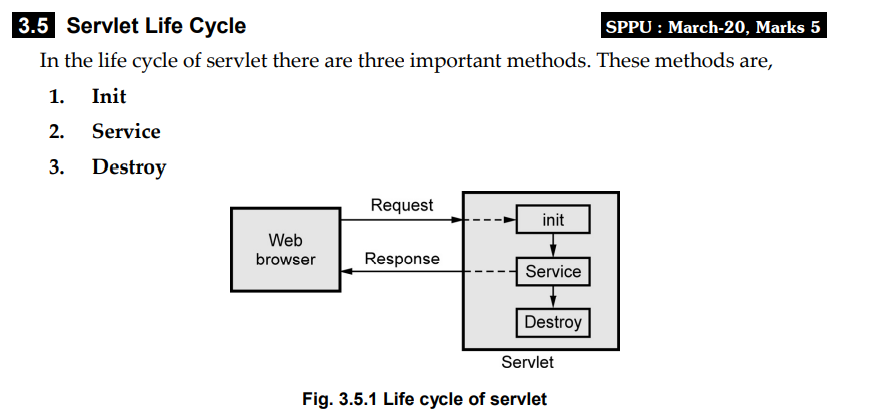
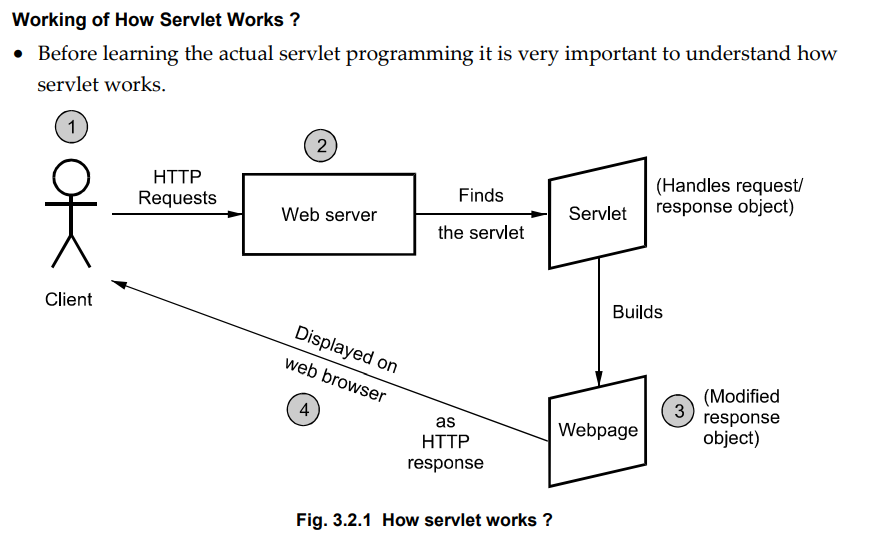
* Data passed as part of URL
* Not secure (visible in URL)
* Used for temporary session tracking

### ✅ Comparison Table (Optional for extra marks):

| **Feature** | **Cookies** | **URL Rewriting** |
| --- | --- | --- |
| Storage | Stored in browser (client-side) | Appended in URL |
| Security | Moderate (can be made HttpOnly) | Low (data exposed in URL) |
| Use Case | Long-term session tracking | Temporary session for small data |
| Limitations | May be disabled by user | URL length limit, exposed in logs |

**Q5. Explain the Servlet architecture with diagram and explain servlet lifecycle. [9]**

**=>**



## ✅ Servlet Architecture

### 📘 Definition:

Servlet architecture is based on the **request-response** model using **HTTP**.  
It is built around the javax.servlet and javax.servlet.http packages and uses **Servlet Container** (e.g., Tomcat) to manage servlet execution.

### 🔧 Components of Servlet Architecture:

1. **Client (Web Browser)**  
   Sends HTTP request using a URL.
2. **Web Server (Servlet Container)**  
   Handles the request, loads servlet, and provides the response.
3. **Servlet Class**  
   Contains logic in doGet() / doPost() to process client request.
4. **Database/Resources**  
   Can be accessed from servlet to retrieve/store data.

### 🖼️ Diagram: Servlet Architecture

arduino

CopyEdit

┌──────────────┐

│ Browser │

└────┬─────────┘

│ (1) HTTP Request

▼

┌─────────────────────┐

│ Web Server / │

│ Servlet Container │

└────────┬────────────┘

│

┌─────▼─────┐

│ Servlet │

└────┬──────┘

│

┌──────▼────────┐

│ JDBC / DB │

└───────────────┘

## ✅ Servlet Lifecycle

### 📘 What is Servlet Lifecycle?

The **Servlet Lifecycle** defines the process through which a servlet is:

1. **Loaded**
2. **Initialized**
3. **Handles requests**
4. **Destroyed**

### 🔁 Servlet Lifecycle Methods:

| **Stage** | **Method** | **Description** |
| --- | --- | --- |
| **1. Initialization** | init() | Called once when servlet is first loaded. Used to initialize resources. |
| **2. Request Handling** | service() → doGet() / doPost() | Called for every client request. Handles business logic. |
| **3. Destruction** | destroy() | Called once when servlet is being unloaded. Used to release resources. |

### 🧾 Lifecycle Flow:

plaintext

CopyEdit

1. Servlet class loaded

2. init() → Initialization

3. service() → handles each request

4. doGet() / doPost() → actual logic

5. destroy() → cleanup during shutdown

### ✅ Example (Code Skeleton):

java

CopyEdit

public class MyServlet extends HttpServlet {

public void init() {

// Initialization code

}

protected void doGet(HttpServletRequest req, HttpServletResponse res)

throws ServletException, IOException {

res.getWriter().println("Response from Servlet");

}

public void destroy() {

// Cleanup code

}

}

### 📝 Conclusion:

* **Servlet architecture** enables dynamic content serving in a web application.
* **Lifecycle** ensures servlets are efficiently created, executed, and destroyed by the container.

**Q7. Explain the Servlet and MySQL database connectivity with example code to display data from employee (emp\_id, emp\_name, emp\_dept) table. [9]**

**=>**

## ✅ Servlet and MySQL Database Connectivity

### 📘 Definition:

Servlets are **Java-based server-side programs** that can interact with databases using **JDBC (Java Database Connectivity)**.  
This connectivity allows dynamic web applications to **store, retrieve, update, or delete data** from a **relational database** such as **MySQL**.

## ✅ Importance of Servlet-DB Connectivity:

* Enables **dynamic data-driven webpages** (e.g., student records, login systems).
* Allows **real-time access** to backend data from user interfaces.
* Common in **enterprise web applications** where a browser communicates with a central database via Servlets.

## ✅ Steps to Connect Servlet with MySQL

1. **Import JDBC and Servlet libraries**  
   java.sql.\*, javax.servlet.\*, javax.servlet.http.\*
2. **Load the MySQL JDBC Driver**

java

CopyEdit

Class.forName("com.mysql.cj.jdbc.Driver");

1. **Establish Connection to MySQL**

java

CopyEdit

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db\_name", "username", "password");

1. **Create Statement & Execute Query**

java

CopyEdit

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM employee");

1. **Process and Display Results**
2. **Close Connection**  
   Always close Connection, Statement, and ResultSet to avoid memory leaks.

## ✅ Example: Display Employee Data

java

CopyEdit

import java.io.\*;

import java.sql.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class EmployeeServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

try {

// Load JDBC driver

Class.forName("com.mysql.cj.jdbc.Driver");

// Connect to MySQL

Connection con = DriverManager.getConnection(

"jdbc:mysql://localhost:3306/college", "root", "root");

// Execute query

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM employee");

// Display data

out.println("<h2>Employee List</h2>");

out.println("<table border='1'>");

out.println("<tr><th>ID</th><th>Name</th><th>Dept</th></tr>");

while (rs.next()) {

out.println("<tr>");

out.println("<td>" + rs.getInt("emp\_id") + "</td>");

out.println("<td>" + rs.getString("emp\_name") + "</td>");

out.println("<td>" + rs.getString("emp\_dept") + "</td>");

out.println("</tr>");

}

out.println("</table>");

// Close connection

con.close();

} catch (Exception e) {

out.println("Error: " + e.getMessage());

}

}

}

## ✅ Supporting Details:

### 🔹 JDBC Driver:

* Required JAR: mysql-connector-j.jar
* Must be placed in the project's WEB-INF/lib or added to the build path.

### 🔹 Deployment:

* Add servlet mapping in web.xml:

xml

CopyEdit

<servlet>

<servlet-name>EmployeeServlet</servlet-name>

<servlet-class>EmployeeServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>EmployeeServlet</servlet-name>

<url-pattern>/viewEmployees</url-pattern>

</servlet-mapping>

### 🔹 Error Handling:

* Always use try-catch for connection code to handle SQL or ClassNotFound exceptions.

## 📝 Conclusion:

Servlets can be connected to a MySQL database using **JDBC**, enabling real-time dynamic applications. The above example demonstrates how to **fetch and display data** from an employee table using a **simple servlet**.

**Q8. Write note on: i) AJAX ii) XML iii) XML transformation [15]**

**=>**

## ✅ (i) AJAX – Asynchronous JavaScript and XML

### 📘 Definition:

AJAX stands for **Asynchronous JavaScript and XML**. It is a **web development technique** used on the **client-side** to **send and receive data from a server asynchronously**, without reloading the entire page.

AJAX is not a programming language — it is a **combination** of:

* HTML & CSS for structure and style
* JavaScript for scripting
* **XMLHttpRequest** object for asynchronous server communication
* Data formats like **XML** or **JSON**

### 🔄 How AJAX Works:

1. User triggers an event (e.g., button click).
2. JavaScript creates an **XMLHttpRequest** object.
3. The request is sent to the server asynchronously.
4. The server processes the request and returns a response.
5. JavaScript processes the response and **updates part of the web page** using DOM.

### 🧾 AJAX Example:

javascript

CopyEdit

var xhttp = new XMLHttpRequest();

xhttp.open("GET", "data.xml", true);

xhttp.onreadystatechange = function() {

if (this.readyState === 4 && this.status === 200) {

document.getElementById("output").innerHTML = this.responseText;

}

};

xhttp.send();

### ✅ Advantages of AJAX:

* **No full-page reload**
* **Faster user interactions**
* **Reduces server load**
* **Improves performance and user experience**
* Seamless **background data processing**

### ✅ Applications of AJAX:

* Real-time form validation
* Live search suggestions (e.g., Google Search box)
* Auto-refreshing feeds (e.g., Facebook/Instagram comments)
* Email systems like Gmail

## ✅ (ii) XML – Extensible Markup Language

### 📘 Definition:

**XML (Extensible Markup Language)** is a markup language that defines **a set of rules for encoding documents** in a format that is **both human-readable and machine-readable**.

* XML is a **platform-independent** and **self-descriptive** language.
* Used for **data storage**, **transport**, and **configuration** across software systems.

### 📦 Characteristics of XML:

* **Custom tag definitions** (unlike HTML)
* Case-sensitive
* Strictly hierarchical tree structure
* Tags must be **properly nested** and **closed**

### 🧾 XML Example:

xml

CopyEdit

<employee>

<emp\_id>101</emp\_id>

<emp\_name>Ravi</emp\_name>

<emp\_dept>IT</emp\_dept>

</employee>

### ✅ Features of XML:

* Supports **data validation** through DTD or XSD
* Easily parsed using DOM or SAX parsers
* Works across multiple platforms and programming languages

### ✅ Applications of XML:

* Web services (SOAP)
* Config files (web.xml, pom.xml, .plist)
* Data exchange between APIs
* Document storage (MS Word/Excel saves as XML internally)

### ✅ Advantages:

* Highly structured and easy to validate
* Extensible: can define your own tags
* Interoperable between diverse systems (Java, .NET, Python)

## ✅ (iii) XML Transformation

### 📘 Definition:

**XML Transformation** is the process of **converting XML data into another format** (like HTML, plain text, or even another XML format) using technologies like **XSLT (Extensible Stylesheet Language Transformations)**.

XML by itself does not present or format the data. Transformation makes it **human-readable** or **suitable for output systems**.

### ⚙️ Key Components:

* **XML Document**: Source structured data
* **XSLT Stylesheet**: Transformation rules (like templates)
* **Output Document**: Result (HTML, text, XML)

### 🧾 XML Transformation Example:

**XML:**

xml

CopyEdit

<student>

<name>Rohit</name>

<course>CS</course>

</student>

**XSLT:**

xml

CopyEdit

<xsl:template match="/student">

<html>

<body>

<h2><xsl:value-of select="name"/></h2>

<p>Course: <xsl:value-of select="course"/></p>

</body>

</html>

</xsl:template>

### ✅ Purpose of XML Transformation:

* To **display XML data as HTML** on web browsers
* To convert data for **cross-platform integration**
* To **filter, sort, or modify** XML content

### ✅ Benefits of XSLT:

* Separation of **data (XML)** from **presentation (XSLT)**
* Dynamic content generation
* Reusable template-based formatting
* Declarative, easy to maintain

### ✅ Applications of XML Transformation:

* Displaying XML data on web pages
* Converting XML into reports (PDF, CSV, Excel)
* Middleware communication formatting
* Generating configuration or migration files

## 📝 Final Summary:

| **Concept** | **Purpose** | **Tool / Technology Used** |
| --- | --- | --- |
| **AJAX** | Asynchronous server interaction without reload | JavaScript, XMLHttpRequest |
| **XML** | Portable structured data format | Tags, DTD/XSD, DOM/SAX |
| **XML Transformation** | Convert XML to human- or machine-readable formats | XSLT (Extensible Stylesheet Language) |

**Q9. What are strengths of XML technology? Explain the need of XML. [5]**

**=>**

## ✅ XML Technology – Strengths and Need

### 📘 What is XML?

**XML (eXtensible Markup Language)** is a platform-independent, text-based format for **storing and exchanging structured data** between systems.  
It is both **human-readable and machine-readable**, and supports **custom tags** for any kind of data.

## ✅ Strengths of XML Technology:

| **Sr.** | **Strength** | **Explanation** |
| --- | --- | --- |
| 1 | **Platform Independent** | Works across all OS and programming languages (Java, Python, .NET, etc.) |
| 2 | **Self-Descriptive Structure** | Tags describe data meaning (e.g., <emp\_name>Rahul</emp\_name>) |
| 3 | **Extensible and Flexible** | Users can define custom tags and structures as per need |
| 4 | **Supports Validation** | Ensures data accuracy via DTD or XSD |
| 5 | **Supports Hierarchical Structure** | Ideal for representing complex/nested data |
| 6 | **Interoperability** | Used as a common data format across different systems and platforms |
| 7 | **Wide Adoption** | Used in Web Services (SOAP), RSS feeds, config files, mobile apps, etc. |

## ✅ Need of XML:

1. **Data Sharing Across Platforms**  
   XML provides a **common format** to exchange data between applications developed in different technologies.
2. **Data Storage and Transport**  
   XML can be used to **store and transmit data** between client and server (e.g., in AJAX, web services).
3. **Web and Enterprise Applications**  
   Widely used in **config files**, **document formats**, and **web APIs**.
4. **Validation and Integrity**  
   Through **DTD/XSD**, XML ensures that the data is well-formed and valid.
5. **Customizable Data Representation**  
   Unlike HTML, XML allows defining your **own tags** as per the application's logic.

### 🧾 Example:

xml

CopyEdit

<employee>

<id>101</id>

<name>Ravi</name>

<dept>IT</dept>

</employee>

## 📝 Conclusion:

XML is essential for building **interoperable, portable, and structured applications**. Its flexibility, self-description, and validation support make it a **backbone of modern web and data systems**.

**Q10. What are DTD’s? Explain how do they work? [5]**

**=>**

## ✅ What are DTDs? (Document Type Definitions)

### 📘 Definition:

**DTD (Document Type Definition)** is a set of rules that defines the **legal building blocks of an XML document**, such as:

* What elements can appear
* The **order and nesting** of elements
* What **attributes** are allowed and their data types

It is used to **validate** whether an XML document is **well-formed** and **valid**.

### 🔍 Why Use DTD?

* Ensures **structure correctness** in XML files.
* Prevents **missing or invalid tags** in documents.
* Helps different systems agree on a **common XML structure** (important in APIs and data exchange).

## 🧠 How DTDs Work:

When an XML document is associated with a DTD:

* The **XML parser reads the DTD**
* Checks that all elements and attributes in the XML file are defined in the DTD
* Verifies **order, occurrence, and nesting** of tags

If everything matches, the XML is called **valid**.

## 📂 Types of DTD:

### 1️⃣ ****Internal DTD**** (Declared inside XML file)

#### 🧾 Example:

xml

CopyEdit

<!DOCTYPE employee [

<!ELEMENT employee (name, dept)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT dept (#PCDATA)>

]>

<employee>

<name>Ravi</name>

<dept>IT</dept>

</employee>

* <!ELEMENT> defines valid XML elements
* #PCDATA means parsed character data (text)

### 2️⃣ ****External DTD**** (Stored in separate .dtd file)

**File: employee.dtd**

dtd

CopyEdit

<!ELEMENT employee (name, dept)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT dept (#PCDATA)>

**File: employee.xml**

xml

CopyEdit

<!DOCTYPE employee SYSTEM "employee.dtd">

<employee>

<name>Ravi</name>

<dept>IT</dept>

</employee>

* SYSTEM refers to the file name or URL containing DTD rules.

## 📈 Features of DTD:

* Supports element ordering ((name, age) means name must come before age)
* Allows repetition using:
  + \* = 0 or more
  + + = 1 or more
  + ? = 0 or 1

## ✅ Advantages of DTD:

| **Advantage** | **Description** |
| --- | --- |
| ✅ Platform Independent | Can be used in any system |
| ✅ Ensures Data Consistency | Validates XML structure |
| ✅ Lightweight | Simpler and faster to parse |
| ✅ Easy to Share | Makes XML interoperable between systems |

## ❌ Limitations of DTD:

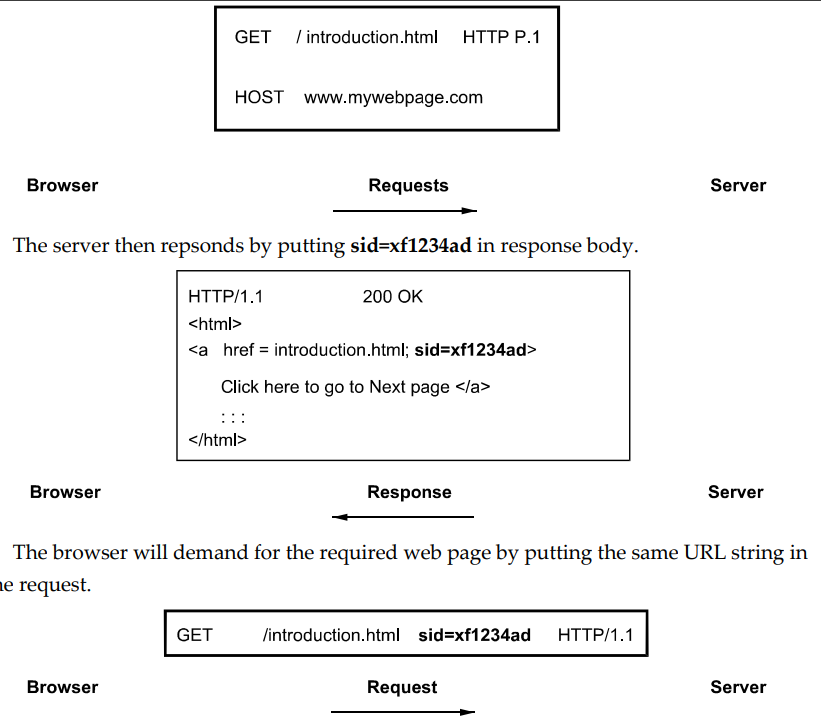
| **Limitation** | **Description** |
| --- | --- |
| ❌ No Data Types | Only text-based validation |
| ❌ No Namespace Support | Cannot distinguish between elements from different vocabularies |
| ❌ Not XML-based Syntax | Uses separate grammar (unlike XML Schema) |

## 📝 Conclusion:

**DTD** plays a crucial role in XML applications by defining what an XML document must look like.  
It is widely used in systems where **data format validation** is critical — like in **web services, configuration files, and APIs**.  
Although newer alternatives like **XML Schema (XSD)** offer more features, DTDs are still used for their **simplicity and speed**.

**Q11. Explain URL writing and cookies in servlet with example. [8]**

**=>**



## ✅ Session Management in Servlets

Since **HTTP is a stateless protocol**, every client request is treated independently.  
To maintain a **user session** (i.e., continue data flow across multiple pages), we use **session tracking techniques** like:

* Cookies
* URL Rewriting
* Hidden Form Fields
* HttpSession (advanced)

In this question, we focus on the **two basic techniques**: **URL Rewriting** and **Cookies**.

## 🔷 (1) URL Rewriting

### 📘 Definition:

**URL rewriting** is a technique to pass **session information** from one servlet to another by **appending data as parameters in the URL**.

It is useful when cookies are **disabled in the browser**.

### 🛠 How it Works:

* The server encodes the session or user data in the **URL string**
* The client clicks the rewritten URL
* The server **reads the data** from the URL query parameters

### 🧾 Example:

**Sending URL with parameter:**

java

CopyEdit

String name = "Ravi";

response.sendRedirect("SecondServlet?username=" + name);

**Receiving parameter in SecondServlet:**

java

CopyEdit

String user = request.getParameter("username");

out.println("Welcome, " + user);

### ✅ Features:

* No need for cookies or client-side storage
* Works well with basic browsers
* Limited by **URL length**

### ❌ Disadvantages:

* Data is visible in the URL (not secure)
* Not suitable for large or sensitive data
* Cannot maintain long-term session easily

## 🔷 (2) Cookies

### 📘 Definition:

A **cookie** is a small piece of data stored on the **client browser** by the server.  
It is automatically sent back to the server **with every HTTP request**.

### 🛠 How Cookies Work:

1. Server creates a cookie and sends it in response
2. Browser stores the cookie
3. On every new request, browser sends the cookie back to the server

### 🧾 Servlet Example:

**Setting a cookie in Servlet:**

java

CopyEdit

Cookie ck = new Cookie("username", "Ravi");

response.addCookie(ck);

**Reading the cookie in another Servlet:**

java

CopyEdit

Cookie[] cookies = request.getCookies();

for (Cookie c : cookies) {

if (c.getName().equals("username")) {

out.println("Welcome, " + c.getValue());

}

}

### ✅ Features:

* Cookies store **persistent** data (can remain until expiration)
* Sent automatically with requests
* Widely supported by all browsers

### ❌ Disadvantages:

* May be disabled by the user
* Small size limit (~4 KB)
* Privacy concerns

## 📊 URL Rewriting vs Cookies

| **Feature** | **URL Rewriting** | **Cookies** |
| --- | --- | --- |
| **Storage** | Data in URL | Data in client browser |
| **Security** | Less secure (visible in URL) | Moderate (can be HttpOnly, Secure) |
| **Size Limit** | Limited by URL length | ~4 KB per cookie |
| **Browser Support** | Works without cookies | Requires cookie support |
| **Data Persistence** | Only for current session | Can be persistent |

## 📝 Conclusion:

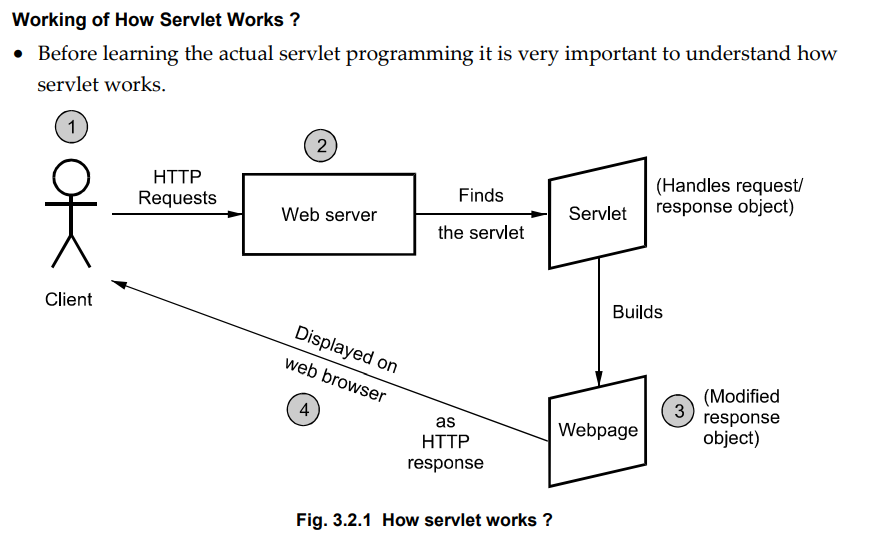
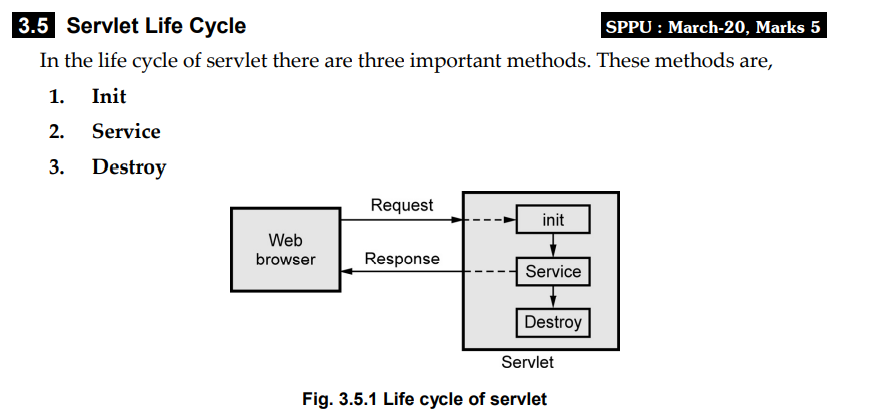
Both **URL rewriting** and **cookies** are widely used techniques in servlet-based web applications to maintain sessions and track users.

* Use **URL rewriting** when cookies are disabled or for quick navigation
* Use **cookies** for persistent user data across multiple sessions

These methods help ensure that web applications provide a **personalized and continuous user experience** despite the stateless nature of HTTP.

**Q12. Explain servlet architecture in detail. [8]**

**=>**



## ✅ Servlet Architecture

### 📘 What is a Servlet?

A **Servlet** is a **Java class** that runs on a **Java-enabled web server** (like Apache Tomcat) and handles **client requests** (usually HTTP), processes them, and returns a **response** (like HTML or JSON).

It follows a **request-response** model and is a core part of **Java EE web applications**.

## ✅ Servlet Architecture Overview

Servlet architecture is built around **4 main components**:

1. **Client (Browser)**
2. **Web Server / Servlet Container**
3. **Servlet Class (Business Logic)**
4. **Database / Backend Resources (Optional)**

These components interact in a specific sequence for processing web requests.

## 🛠️ Detailed Flow of Servlet Architecture:

### 🔁 Step-by-step Request Processing:

1. **Client Request**:
   * A web browser or client sends an **HTTP request** to the server.
   * Example: A user visits http://example.com/login.
2. **Web Server / Servlet Container**:
   * The server receives the request and passes it to the **Servlet Container** (like Tomcat).
   * The container checks **web.xml** (or annotations) to locate the matching servlet.
3. **Servlet Lifecycle Begins** (if not already loaded):
   * init() is called once when the servlet is first loaded.
   * service() method is called for each request.
   * doGet() or doPost() is triggered depending on request type.
4. **Processing Business Logic**:
   * Servlet processes user data, interacts with a **database** (if needed), performs logic.
5. **Response Generation**:
   * Servlet writes a response using response.getWriter().print().
   * Data is returned as **HTML**, **JSON**, or other formats.
6. **Client Receives Response**:
   * The browser displays the response to the user.
7. **destroy()**:
   * When server shuts down or servlet is no longer needed, the container calls destroy() to release resources.

## 📦 Core Components of Servlet Architecture:

| **Component** | **Description** |
| --- | --- |
| **Client** | Web browser (Chrome, Firefox) |
| **Web Server** | Hosts the servlet (e.g., Apache Tomcat) |
| **Servlet Container** | Manages servlet lifecycle, routing, and communication |
| **Servlet Class** | Java class containing doGet()/doPost() logic |
| **Request/Response Objects** | Instances of HttpServletRequest and HttpServletResponse used to read input and write output |

## 🖼️ Text-based Diagram (for writing in exam):

pgsql

CopyEdit

Client (Browser)

│

▼

Web Server / Servlet Container

│

▼

Servlet (doGet/doPost)

│

▼

[Business Logic / Database Access]

│

▼

Http Response → Back to Client

## ✅ Role of the Servlet Container:

The **Servlet Container** is a part of the web server responsible for:

* **Loading** servlet classes
* **Managing lifecycle** (init(), service(), destroy())
* **Thread management** for concurrent requests
* **Mapping URLs to servlets**
* **Handling HTTP protocol tasks**

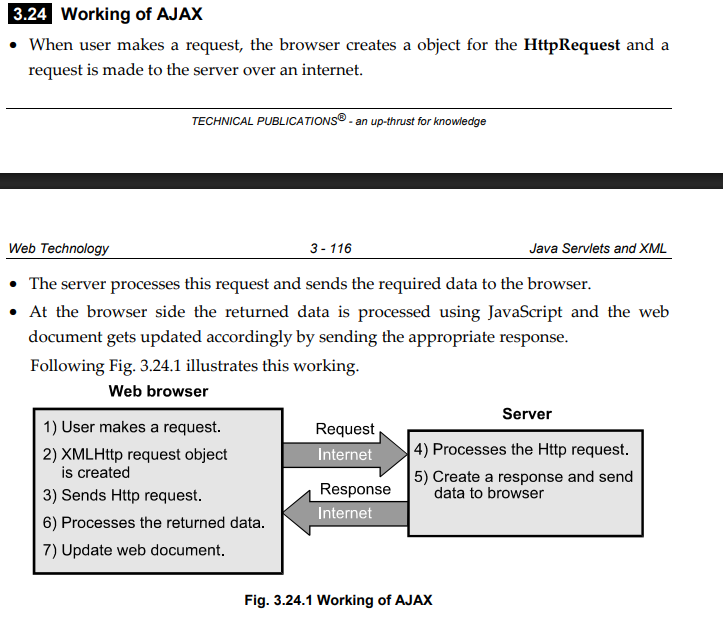
Examples: **Apache Tomcat**, **Jetty**, **GlassFish**

## 📝 Conclusion:

Servlet architecture is the backbone of Java-based web development.  
It allows efficient processing of client requests using a **modular**, **scalable**, and **platform-independent** structure.  
The Servlet Container plays a critical role in managing servlet classes and enabling **dynamic web application functionality**.

**Q13. What do you understand by AJAX?. Explain it. [6]**

**=>**



## ✅ What is AJAX?

### 📘 Definition:

**AJAX** stands for **Asynchronous JavaScript and XML**.  
It is a **web development technique** that allows **data to be sent and received from a server in the background**, **without reloading the entire webpage**.

AJAX enables the creation of **dynamic, fast, and responsive web applications** by updating parts of a webpage asynchronously.

## 🔁 How AJAX Works:

AJAX works using the **XMLHttpRequest** object in JavaScript.

### 🔄 Flow:

1. User performs an action (e.g., button click)
2. JavaScript creates an XMLHttpRequest
3. Request is sent to the server (in background)
4. Server processes the request and returns data (XML, JSON, etc.)
5. JavaScript uses the response to update part of the webpage **without full reload**

## 🧾 AJAX Example:

html

CopyEdit

<!DOCTYPE html>

<html>

<head>

<script>

function loadData() {

var xhr = new XMLHttpRequest();

xhr.open("GET", "data.txt", true);

xhr.onreadystatechange = function () {

if (xhr.readyState == 4 && xhr.status == 200) {

document.getElementById("output").innerHTML = xhr.responseText;

}

};

xhr.send();

}

</script>

</head>

<body>

<button onclick="loadData()">Load Data</button>

<div id="output"></div>

</body>

</html>

✅ When the button is clicked, AJAX fetches data.txt and shows its content inside the output <div> — **without page reload**.

## ⚙️ Key Technologies Behind AJAX:

* **HTML/CSS** → for structure and style
* **JavaScript** → for logic and dynamic behavior
* **XMLHttpRequest** → core for making background requests
* **XML/JSON** → format for server responses (can be plain text too)

## 🌟 Features of AJAX:

* Asynchronous data transfer
* Fast response to user input
* Partial page updates (no full reload)
* Supports various formats: XML, JSON, HTML, Text

## 💼 Real-Life Applications:

| **Feature** | **Examples** |
| --- | --- |
| Search Suggestions | Google Instant Search |
| Live Data Updates | Stock price trackers |
| Auto-Refresh Content | Facebook comments, Instagram |
| Form Validation | Gmail login / registration |
| Background Data Load | YouTube comments, tweets |

## ✅ Advantages of AJAX:

* **Better User Experience**
* **Reduces Bandwidth Usage** (no reload of full page)
* **Fast and Lightweight**
* **Cross-platform support** (supported by all modern browsers)

## ❌ Limitations of AJAX:

* Can make debugging more complex
* JavaScript must be enabled in browser
* Not ideal for pages that need to be indexed by search engines
* Increases reliance on client-side scripting

## 📝 Conclusion:

AJAX is a powerful tool that allows modern web applications to behave like desktop applications — **faster, responsive, and interactive**. It helps deliver **rich user experiences** and is widely used in modern websites and SPAs (Single Page Applications).

**Q14. Write a servlet which accepts two numbers using POST method and display the maximum number. [5]**

**=>**

### 🧾 HTML Form to Accept Input:

(Save as index.html)

html

CopyEdit

<!DOCTYPE html>

<html>

<head><title>Find Maximum</title></head>

<body>

<form action="MaxServlet" method="post">

Enter First Number: <input type="text" name="num1"><br>

Enter Second Number: <input type="text" name="num2"><br>

<input type="submit" value="Find Maximum">

</form>

</body>

</html>

### 🧾 Servlet Code: MaxServlet.java

java

CopyEdit

import java.io.\*;

import javax.servlet.\*;

import javax.servlet.http.\*;

public class MaxServlet extends HttpServlet {

public void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

// Read numbers from form

int num1 = Integer.parseInt(request.getParameter("num1"));

int num2 = Integer.parseInt(request.getParameter("num2"));

// Find maximum

int max = (num1 > num2) ? num1 : num2;

// Display result

out.println("<h2>Maximum Number is: " + max + "</h2>");

}

}

### ✅ Deployment Configuration (web.xml)

(If annotations are not used)

xml

CopyEdit

<servlet>

<servlet-name>MaxServlet</servlet-name>

<servlet-class>MaxServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>MaxServlet</servlet-name>

<url-pattern>/MaxServlet</url-pattern>

</servlet-mapping>

## 📝 Conclusion:

This servlet demonstrates the use of **doPost()** to handle user input from an HTML form, perform simple logic, and output a result dynamically — all part of basic Java Servlet operations.

**Q15. Explain the following: i) Process of transforming XML document. ii) HTTP session. [8]**

**=>**

## ✅ Part i) Process of Transforming XML Document

### 📘 What is XML Transformation?

**XML Transformation** is the process of converting an XML document into another format such as:

* **HTML**
* **Plain Text**
* **Another XML format**

This is usually done using **XSLT** (Extensible Stylesheet Language Transformations), a special language for transforming XML documents.

### 🛠 Tools Involved in Transformation:

| **Component** | **Role** |
| --- | --- |
| **XML Document** | The original structured data |
| **XSLT File** | Defines the transformation rules |
| **XSLT Processor** | Engine that applies rules and generates output |

### 🧾 Example: Transform XML to HTML

**XML Document (student.xml):**

xml

CopyEdit

<student>

<name>Ravi</name>

<course>CS</course>

</student>

**XSLT File (student.xsl):**

xml

CopyEdit

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/student">

<html><body>

<h2><xsl:value-of select="name"/></h2>

<p>Course: <xsl:value-of select="course"/></p>

</body></html>

</xsl:template>

</xsl:stylesheet>

### 🔁 Steps of Transformation:

1. Load the XML document.
2. Link the XSLT file using:

xml

CopyEdit

<?xml-stylesheet type="text/xsl" href="student.xsl"?>

1. Use a browser or XSLT processor to render the result.

### ✅ Benefits:

* Converts machine-readable XML into human-readable HTML.
* Customizable views for the same data source.
* Used in reporting systems, data integration, and APIs.

## ✅ Part ii) HTTP Session

### 📘 Definition:

An **HTTP Session** is a server-side mechanism to **maintain user data across multiple requests** in a web application.

Since HTTP is **stateless**, every client request is independent. A session allows web applications to **"remember" the user** between interactions.

### 🛠 Working of HTTP Session:

1. When a client sends a request, the server generates a **unique session ID**.
2. This ID is sent to the client via a **cookie** or **URL rewriting**.
3. The session data is stored on the server using the **HttpSession object**.
4. On every next request, the session ID is sent back to the server.
5. Server uses the ID to **retrieve and manage session-specific data**.

### 🧾 Example: Storing and retrieving session data

java

CopyEdit

// Create or retrieve session

HttpSession session = request.getSession();

// Store data

session.setAttribute("username", "Rahul");

// Retrieve data

String user = (String) session.getAttribute("username");

out.println("Welcome, " + user);

### ✅ Features of HttpSession:

* **Automatic session tracking**
* Can store **objects**, not just strings
* Sessions can be **timed out** after inactivity
* Works using **cookies** or **URL rewriting**

### ❌ Limitations:

* Server memory usage increases with more sessions.
* Should not be used to store sensitive data unless securely managed.

## 📝 Conclusion:

* **XML Transformation** allows us to convert structured data into readable formats using XSLT, crucial for web output and integration.
* **HTTP Session** ensures consistent and stateful interactions between users and servers, a critical part of modern dynamic web applications.

**Q18. Explain XML with respect to structure, declaration syntax, namespace. [8]**

**=>**

## ✅ XML (eXtensible Markup Language)

XML is a **platform-independent**, **self-descriptive**, and **structured** language used for **storing and transporting data**.  
Unlike HTML (which defines display), XML is focused on the **structure and meaning of data**.

## ✅ A) XML Structure

### 📘 Definition:

XML uses a **tree-like hierarchical structure** to represent data using **user-defined tags**.

### 🔑 Basic Rules:

* Every tag must be properly **opened and closed**
* Tags must be **case-sensitive**
* XML must have a **single root element**
* Elements must be **properly nested**
* Attribute values must be in **quotes**

### 🧾 Example:

xml

CopyEdit

<employee>

<emp\_id>101</emp\_id>

<emp\_name>Ravi</emp\_name>

<emp\_dept>IT</emp\_dept>

</employee>

### 📌 Explanation:

* <employee> is the **root element**
* Inside it, there are **child elements** like <emp\_id>, <emp\_name>, etc.
* This structure makes XML **machine-readable** and **easy to validate**

## ✅ B) XML Declaration Syntax

The **XML declaration** is optional but recommended and appears at the top of the XML file.

### 🧾 Syntax:

xml

CopyEdit

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

### 📘 Parts of the declaration:

| **Attribute** | **Description** |
| --- | --- |
| version | Specifies the XML version (usually 1.0) |
| encoding | Defines character encoding (UTF-8, ISO-8859-1, etc.) |

### ✅ Importance:

* Ensures that XML processors interpret the file correctly.
* Encoding is crucial for multi-language support.

## ✅ C) XML Namespace

### 📘 What is a Namespace?

In XML, a **namespace** is used to **avoid element name conflicts** when combining documents from different XML vocabularies (e.g., SVG and XHTML).

### 🔑 Syntax:

xml

CopyEdit

xmlns:prefix="URI"

### 🧾 Example:

xml

CopyEdit

<bookstore xmlns:bk="http://example.com/books">

<bk:book>

<bk:title>XML Basics</bk:title>

<bk:author>John</bk:author>

</bk:book>

</bookstore>

### 📌 Explanation:

* xmlns:bk declares a **namespace prefix** bk associated with a **URI**.
* All elements prefixed with bk: are identified as part of the books namespace.

### ✅ Why Namespaces Are Important:

| **Feature** | **Benefit** |
| --- | --- |
| Prevents name conflicts | Two XML files can use the same tag name safely |
| Enables data merging | Common in web services and XML APIs |
| Improves clarity | Tags become more descriptive with prefixes |

**Q19. What is difference between server side scripting language and client side scripting language. [5]**

**=>**

## ✅ What is Scripting?

Scripting refers to writing code that automates tasks on **web pages** or **servers**.  
It is mainly classified into two types:

1. **Client-Side Scripting**
2. **Server-Side Scripting**

## 🔹 1) Client-Side Scripting

### 📘 Definition:

Client-side scripting refers to scripts that are **executed on the user's browser** (client device), after the web page is loaded.

### ✅ Examples:

* **JavaScript**
* **HTML5**
* **CSS**
* **VBScript** (obsolete)

### ✅ Purpose:

* Handles **user interaction**
* **Validates forms** before submission
* Dynamically **updates content** without reloading (via AJAX)

## 🔸 2) Server-Side Scripting

### 📘 Definition:

Server-side scripting refers to code that is **executed on the web server** before the page is sent to the client’s browser.

### ✅ Examples:

* **PHP**
* **JSP (Java Server Pages)**
* **ASP.NET**
* **Python (Django, Flask)**
* **Node.js**

### ✅ Purpose:

* **Processes form data**
* **Connects to databases**
* Handles **authentication**, **sessions**, and **dynamic page generation**

## 📊 Difference Between Client-Side and Server-Side Scripting

| **Feature** | **Client-Side Scripting** | **Server-Side Scripting** |
| --- | --- | --- |
| **Execution** | In browser (on user's device) | On web server |
| **Speed** | Fast (no server call needed) | Slower (needs server processing) |
| **Security** | Less secure (code visible) | More secure (code hidden) |
| **Access to DB** | No | Yes |
| **Examples** | JavaScript, HTML, CSS | PHP, JSP, ASP.NET, Node.js |

**Q21. Explain DTD in XML with schemes, elements & attributes. [9]**

**=>**

## ✅ What is DTD in XML?

### 📘 Definition:

**DTD (Document Type Definition)** defines the **structure**, **legal elements**, and **attributes** of an XML document.  
It acts like a **blueprint or grammar** that ensures an XML file is **valid** and follows a specific format.

If an XML file follows the rules defined in a DTD, it is said to be **valid XML**.

## 🔧 Purpose of DTD:

* To define allowed **elements and their nesting**
* To define **attributes** and data types
* To **validate XML documents**
* To promote **data consistency and sharing**

## ✅ Types of DTD Schemes

DTD can be declared in **two ways**:

### 🔹 1. Internal DTD

Defined **within** the XML document using the <!DOCTYPE> declaration.

#### 🧾 Example:

xml

CopyEdit

<!DOCTYPE student [

<!ELEMENT student (name, age)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT age (#PCDATA)>

]>

<student>

<name>Ravi</name>

<age>21</age>

</student>

### 🔹 2. External DTD

Defined in a **separate file** (with .dtd extension) and linked to XML.

#### ✅ employee.dtd:

dtd

CopyEdit

<!ELEMENT employee (emp\_id, emp\_name, emp\_dept)>

<!ELEMENT emp\_id (#PCDATA)>

<!ELEMENT emp\_name (#PCDATA)>

<!ELEMENT emp\_dept (#PCDATA)>

#### ✅ employee.xml:

xml

CopyEdit

<!DOCTYPE employee SYSTEM "employee.dtd">

<employee>

<emp\_id>101</emp\_id>

<emp\_name>Ravi</emp\_name>

<emp\_dept>IT</emp\_dept>

</employee>

## ✅ Declaring Elements in DTD

Elements are declared using <!ELEMENT> keyword:

dtd

CopyEdit

<!ELEMENT tag-name (child-elements)>

<!ELEMENT tag-name (#PCDATA)>

| **Symbol** | **Meaning** |
| --- | --- |
| #PCDATA | Parsed character data (text) |
| \* | 0 or more occurrences |
| + | 1 or more occurrences |
| ? | 0 or 1 occurrence |

### 🔹 Example: Element Declaration

dtd

CopyEdit

<!ELEMENT book (title, author)>

<!ELEMENT title (#PCDATA)>

<!ELEMENT author (#PCDATA)>

## ✅ Declaring Attributes in DTD

Attributes are declared using <!ATTLIST> keyword.

### 🔹 Syntax:

dtd

CopyEdit

<!ATTLIST element-name attribute-name type default>

| **Attribute Type** | **Meaning** |
| --- | --- |
| CDATA | Character data |
| ID | Unique identifier |
| IDREF | Reference to another ID |
| ENUMERATION | A set of allowed values |

### 🔹 Attribute Defaults:

| **Default Type** | **Description** |
| --- | --- |
| #REQUIRED | Must be provided in XML |
| #IMPLIED | Optional |
| #FIXED | Fixed value for attribute |
| "value" | Default value if not provided |

### 🔹 Example: Attribute Declaration

dtd

CopyEdit

<!ELEMENT book (#PCDATA)>

<!ATTLIST book category CDATA #REQUIRED>

### 🧾 XML using attribute:

xml

CopyEdit

<book category="Programming">Java Basics</book>

## ✅ Why Use DTD?

* **Ensures validity** of XML documents
* Allows **reusability** of structure across multiple documents
* Helps in **data exchange**, especially in **web services** and **APIs**
* Supports **document verification** before processing

**Q22. What is session? How cookies & URL rewriting for session management in servlet. [9]**

**=>**

## ✅ What is a Session?

### 📘 Definition:

A **session** is a way to **maintain data (state)** between multiple requests made by the same user to a web server.

HTTP is a **stateless protocol**, meaning it doesn't remember a user between requests. A **session** allows a server to “remember” the user and maintain continuity.

### 🔁 Example Use Cases of Sessions:

* Login authentication
* Shopping cart functionality
* User-specific dashboard
* Temporary storage of user actions

## ✅ How Sessions are Maintained in Servlets?

There are several session tracking techniques. Two important ones covered here:

1. **Cookies**
2. **URL Rewriting**

## 🔸 1. Session Management Using Cookies

### 📘 What is a Cookie?

A **cookie** is a small piece of data stored on the **client’s browser**, sent from the server.  
The browser sends it back to the server **with every request**, allowing session data to be tracked.

### 🧾 Example in Servlet:

**Set a cookie in one servlet:**

java

CopyEdit

Cookie ck = new Cookie("username", "Ravi");

response.addCookie(ck);

**Retrieve it in another servlet:**

java

CopyEdit

Cookie[] cookies = request.getCookies();

for (Cookie c : cookies) {

if (c.getName().equals("username")) {

out.println("Welcome " + c.getValue());

}

}

### ✅ Advantages of Cookies:

* Simple to use
* Supported by all modern browsers
* Useful for small data storage (4 KB)

### ❌ Limitations:

* Can be disabled by user
* Not secure (unless encrypted or marked HttpOnly)
* Limited data size

## 🔸 2. Session Management Using URL Rewriting

### 📘 What is URL Rewriting?

URL rewriting is a technique where **session information is appended to the URL** as query parameters, especially useful when cookies are disabled.

### 🧾 Example:

**Sending session data via URL:**

java

CopyEdit

String username = "Ravi";

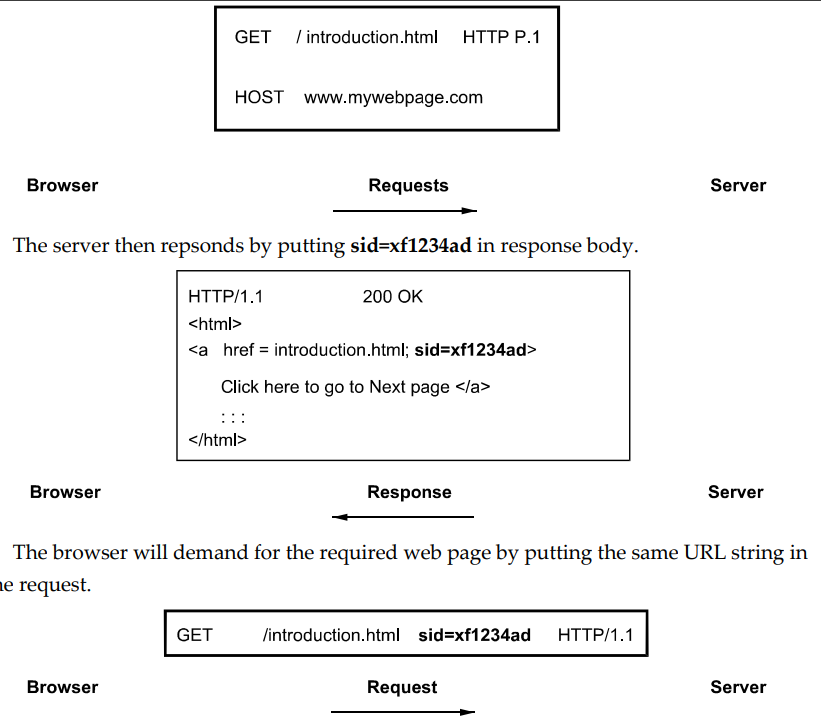
response.sendRedirect("welcome.jsp?user=" + username);

**Retrieving the data in welcome.jsp:**

jsp

CopyEdit

<%= request.getParameter("user") %>



### ✅ Advantages:

* Works even if cookies are disabled
* Simple for small session data
* No need for client-side storage

### ❌ Limitations:

* Visible in the URL (not secure)
* Limited data size
* Cannot bookmark dynamic session data

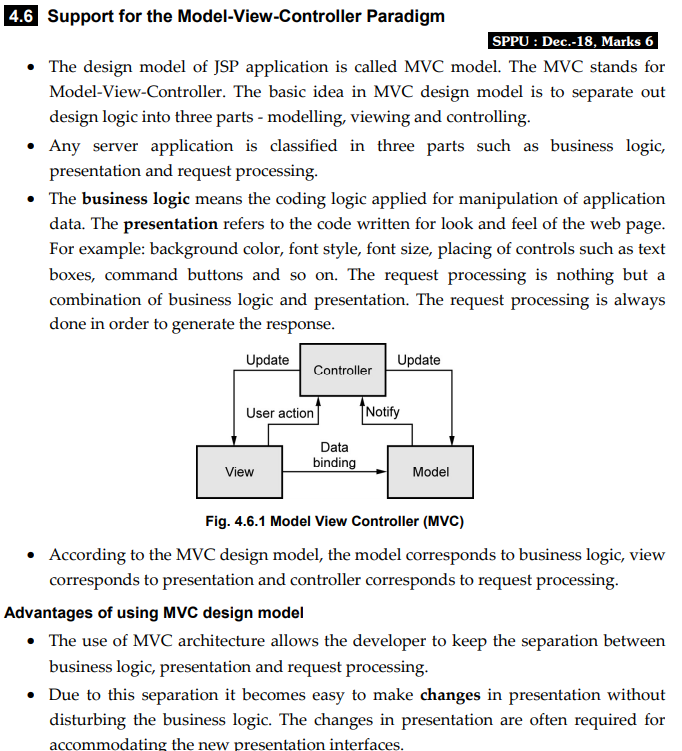
## 🔍 Cookies vs URL Rewriting

| **Feature** | **Cookies** | **URL Rewriting** |
| --- | --- | --- |
| **Storage** | Client browser | Inside the URL |
| **Security** | Moderate | Less secure (visible in URL) |
| **Data Size Limit** | Around 4 KB | Limited by URL length |
| **Browser Support** | May be disabled | Works without cookies |
| **Persistence** | Can be persistent (based on maxAge) | Temporary (per request/session) |

**UNIT 4**

**Q1. Explain the JSP support for MVC paradigm. [8]**

**=>\**



## ✅ JSP Support for MVC Paradigm

### 📘 What is MVC?

**MVC (Model-View-Controller)** is a **design pattern** used in web applications to **separate concerns**:

* **Model**: Business logic and data
* **View**: User Interface
* **Controller**: Request processing and navigation control

## ✅ MVC in JSP-based Applications

In Java web applications, the MVC architecture is commonly implemented using:

| **MVC Component** | **Technology Example** |
| --- | --- |
| **Model** | JavaBeans, JDBC, POJO Classes |
| **View** | **JSP (JavaServer Pages)** |
| **Controller** | **Servlet** |

### 🖼️ MVC Flow using JSP and Servlet:

scss

CopyEdit

Client (Browser)

│

▼

Controller (Servlet) ──> Model (JavaBean/DB logic)

│

▼

View (JSP) ←─── Sends processed data

### 🔄 Step-by-Step Explanation:

1. **Client Request**
   * A user sends a request (e.g., login form submission)
2. **Controller (Servlet)**
   * Servlet receives the request
   * It processes or forwards the request to the **Model**
3. **Model (JavaBean or JDBC code)**
   * Business logic executes (e.g., database query, validation)
   * Result is stored in request or session scope
4. **View (JSP)**
   * JSP accesses the data using Expression Language (EL) or JSTL
   * Generates HTML response for the user

## ✅ Role of JSP in MVC (View Layer)

* **JSP acts as the View** and is responsible for **displaying data** to the user.
* It should **not contain business logic** or request processing logic.
* JSP gets data from the Model via request attributes.

### 🧾 Example:

**Controller (Servlet):**

java

CopyEdit

String username = request.getParameter("username");

request.setAttribute("uname", username);

RequestDispatcher rd = request.getRequestDispatcher("welcome.jsp");

rd.forward(request, response);

**View (JSP):**

jsp

CopyEdit

<h2>Welcome, ${uname}!</h2>

JSP uses **EL (Expression Language)** to access data from the request scope.

## ✅ Benefits of MVC in JSP-based Applications:

| **Advantage** | **Description** |
| --- | --- |
| **Separation of Concerns** | Cleaner code; UI, logic, and flow are independent |
| **Maintainability** | Easy to debug and update |
| **Reusability** | Components can be reused across projects |
| **Scalability** | Easy to extend as application grows |
| **Testability** | Logic can be tested independently of UI |

**Q2. Explain struts framework with respect to architecture, actions, interceptors & exception handling. [9]**

**=>**

**Q3. Explain JSP lifecycle. Differentiate JSP Vs Servlet. (Min.04). [9]**

**=>**

**Q4. Explain the concept of web services. Explain in brief WSDL & SOAP. [8]**

**=>**

**Q5. Explain the concept of JSP with syntax and sample example. Explain the analogy of JSP and Servlets. [9]**

**=>**

**Q6. Explain the concept of WSDL and SOAP. [8]**

**=>**

**Q7. Explain JSP with support for Model View Controller. [9]**

**=>**

**Q8. Explain the concept of struts with architecture, actions, interceptors and exception handling. [8]**

**=>**

**Q9. List & describe important inceptors provided by strut framework. [5]**

**=>**

**Q10. Identify & justify the benefits of using Web Services. [5]**

**=>**

**Q11. Explain JSP life cycle with diagram. [7]**

**=>**

**Q12. What is JSP? Enlist advantages of JSP over servlet? [5]**

**=>**

**Q13. What is WSDL and SOAP? Explain WSDL in detail. [5]**

**=>**

**Q14. Draw and explain MVC architecture for developing web application.[7]**

**=>**

**Q15. Write advantages of JSP over servlet and explain lifecycle of JSP. [8]**

**=>**

**Q16. Explain the Strut architecture with neat diagram and also explain the benefits of Strut. [9]**

**=>**

**Q17. Write a JSP program to demonstrate use of page directive, Scriplet Expression and Comment. [9]**

**=>**

**Q18. Write the benefits of Web services and explain SOAP, Rest and UDDI.[8]**

**=>**

**Q19. Explain life cycle of JSP. Write advantages of JSP over servlet. [9]**

**=>**

**Q20. Write a short notes on: i) SOAP ii) WSDL [8]**

**=>**

**Q21. Explain streets framework with its components. Also explain interceptors. [9]**

**=>**

**Q22. Explain JSP support for MVC i.e. model, view controller for developing web application. [8]**

**=>**

**UNIT 5**

**Q1. Explain various types of Arrays in PHP. Explain each with example code. [9]**

**=>**

**Q2. Explain the following:**

**i) WAP & WML [4]**

**ii) C# Vs Java [5]**

**=>**

**Q3. Explain the PHP with MySQL using example. [6]**

**=>**

**Q4. Write note on: i) Session tracking in PHP. ii) NET framework iii) Node JS [12]**

**=>**

**Q5. Explain different types of arrays in PHP with example [9]**

**=>**

**Q6. Write note on: i) ASP.NET ii) Node JS [9]**

**=>**

**Q7. Write PHP code to connect MYSQL database to display records from the table [9]**

**Q8. Explain the concepts of WAP, WML and .NET framework. [9]**

**Q9. What is WAP? Explain components of WAP architecture in detail. [8]**

**Q10. What is multidimentional arrays in PHP? Explain it with simple PHP code. [5]**

**Q11. Explain overview of node JS. [5]**

**Q12. Explain how cookies and session are used for session management in PHP. [8]**

**Q13. What is WML? Explain WML elements. [5]**

**Q14. Explain in brief overview of ASP. NET. [5]**

**Q15. Explain the following with respects to PhP. [9]**

**i) Arrays**

**ii) Function**

**iii) Control statements in PhP**

**Q16. How does this array work in PhP? Explain with example. [9]**

**Q17. Explain object oriented way to connect MYSQL database with PhP. [9]**

**Q18. Draw and explain .NET framework with CLR, CLI. [9]**

**Q19. Identify and explain steps involved in connecting to mySQL with PHP. [9]**

**Q20. Write short notes on: i) Overview of ASP. NET ii) Overview of C# [9]**

**Q21. Explain in detail WAP Architecture & WML. [9]**

**Q22. Explain functions in PHP with example & session management. [9]**

**UNIT 6**

**Q1. Explain Ruby with its advantages. Explain control statements in Ruby. [10]**

**Q2. Explain EJB concept & five basic example of using EJB. [7]**

**Q3. Explain the arrays in Ruby. Explain Rails with AJAX. [10]**

**Q4. Explain Document Request in Rails. [4]**

**Q5. Explain advantages of Ruby and Rails. [3]**

**Q6. Explain scalar types, operations and pattern matching in Ruby. [9]**

**Q7. Explain documents requests and processing forms in Rails. [8]**

**Q8. Explain the concept of classes and arrays in Ruby. [9]**

**Q9. Explain concepts of Rails with AJAX and EJB. [8]**

**Q10. Explain how multiple selection constructs are implemented in Ruby. [7]**

**Q11. Explain Rails with AJAX in detail. [5]**

**Q12. Draw & explain the role of EJB container in Enterprise applications. [5]**

**Q13. Explain how to write the methods and call the method in RUBY with example. [7] Q14. What are the difference between java beans and EJB? [5]**

**Q15. What is Ruby programming language? List some features of Ruby. [5]**

**Q16. Explain scalar types and their operations in Ruby. [9]**

**Q17. What are the positive aspects of Rails, explain with example. [9]**

**Q18. Write short note on: i) Rails with AJAX ii) WAP and WML [9]**

**Q19. What is EJB? Explain types of EJBs. [9]**

**Q20. Explain classes and objects in Ruby with appropriate examples. [9]**

**Q21. Introduce the concept of Rails application. Describe layouts & stylesheet in Rail. [8]**

**Q22. Explain the scalar types & their operations in Ruby. [8]**

**Q23. Explain Architecture of EJB & explain types of EJB in detail. [9]**