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.

3.5 Servlet Life Cycle **SPPU : March-20, Marks 5**

In the life cycle of servlet there are three important methods. These methods are,

1. **Init**
2. **Service**
3. **Destroy**

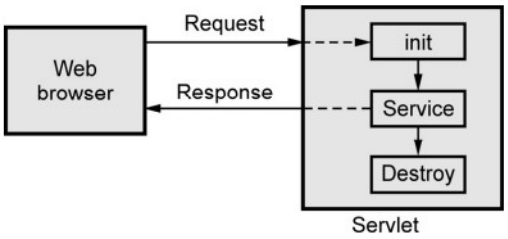


Fig. 3.5.1 Life cycle of servlet

📌 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]

=>

Working of How Servlet Works ?

- Before learning the actual servlet programming it is very important to understand how servlet works.

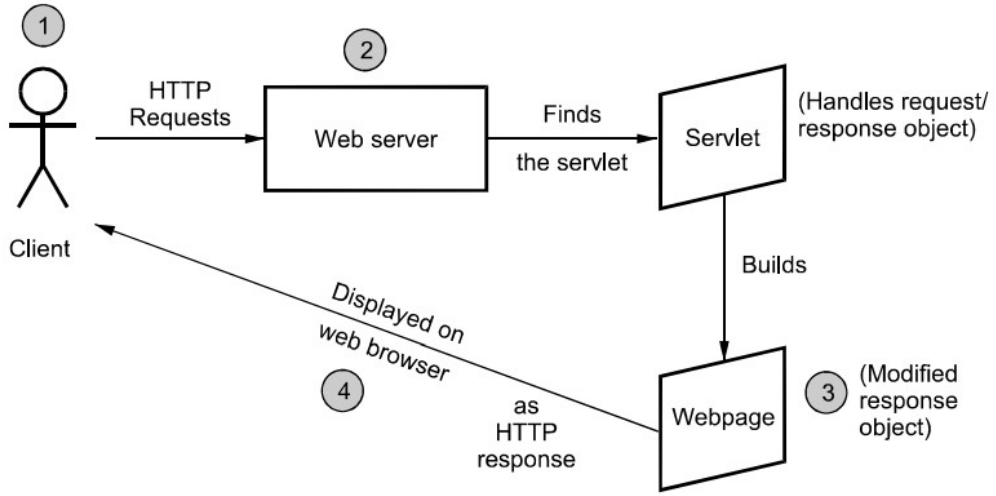


Fig. 3.2.1 How servlet works ?

3.5 Servlet Life Cycle

SPPU : March-20, Marks 5

In the life cycle of servlet there are three important methods. These methods are,

1. **Init**
2. **Service**
3. **Destroy**

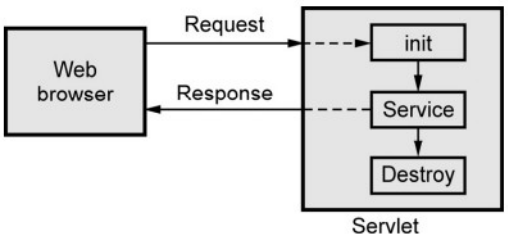


Fig. 3.5.1 Life cycle of servlet

✓ Servlet Architecture

📄 Definition:

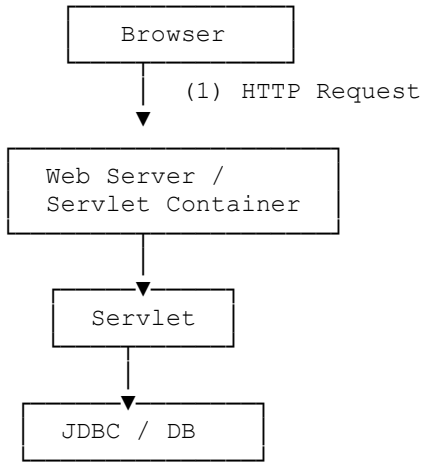
Servlet architecture is based on the **request-response** model using **HTTP**.
It is built around the `javax.servlet` and `javax.servlet.http` packages and uses **Servlet Container** (e.g., Tomcat) to manage servlet execution.

🔧 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
```



✅ 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");
```

3. **Establish Connection to MySQL**

```
java
CopyEdit

Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/db_name",
"username", "password");
```

4. **Create Statement & Execute Query**

```
java
CopyEdit
```

```
        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT * FROM employee");
```

5. **Process and Display Results**
6. **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

Concept	Purpose	Tool / Technology Used
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:

1 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)

---

2 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

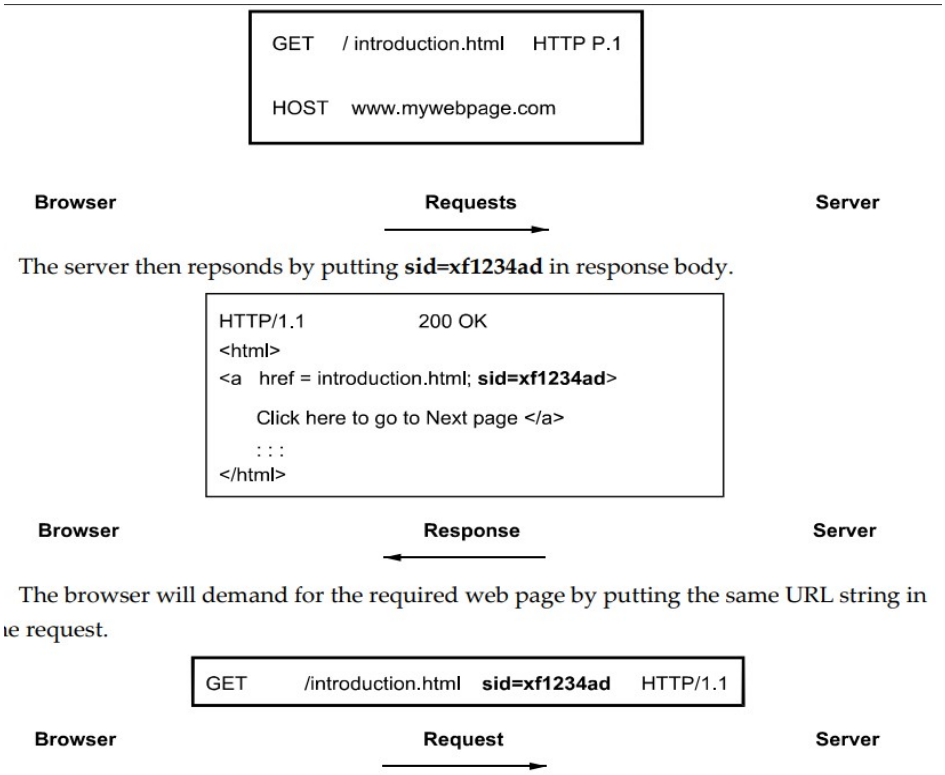
Limitation	Description
❌ No Namespace Support	Cannot distinguish between elements from different vocabularies
❌ Not XML-based Syntax	Uses separate grammar (unlike XML Schema)

📝 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]

=>

3.5 Servlet Life Cycle

SPPU : March-20, Marks 5

In the life cycle of servlet there are three important methods. These methods are,

1. **Init**
2. **Service**
3. **Destroy**

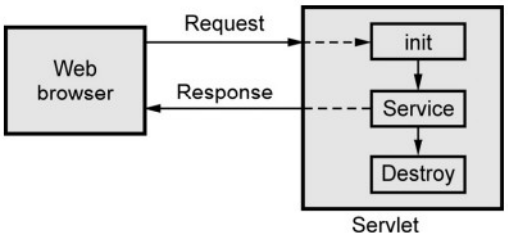


Fig. 3.5.1 Life cycle of servlet

Working of How Servlet Works ?

- Before learning the actual servlet programming it is very important to understand how servlet works.

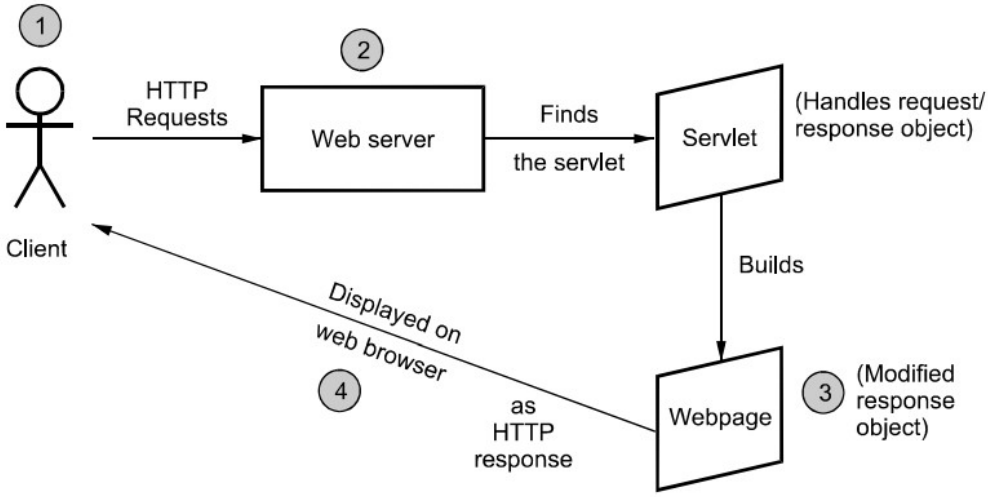


Fig. 3.2.1 How servlet works ?

✓ 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]

=>

3.24 Working of AJAX

- When user makes a request, the browser creates a object for the **HttpRequest** and a request is made to the server over an internet.

- The server processes this request and sends the required data to the browser.
- At the browser side the returned data is processed using JavaScript and the web document gets updated accordingly by sending the appropriate response.

Following Fig. 3.24.1 illustrates this working.

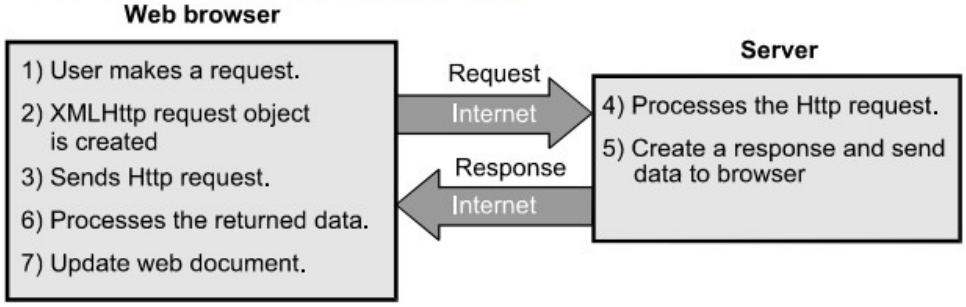


Fig. 3.24.1 Working of AJAX

✓ 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"?>
```

3. 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:

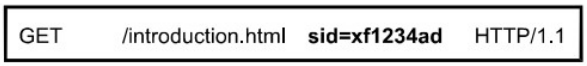
```
jsp
CopyEdit
<%= request.getParameter("user") %>
```

Browser Requests Server

The server then repsonds by putting **sid=xf1234ad** in response body.

Browser Response Server

The browser will demand for the required web page by putting the same URL string in ıe request.



Browser Request Server

✅ 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

Feature	Cookies	URL Rewriting
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]

=>\

4.6 Support for the Model-View-Controller Paradigm

SPPU : Dec.-18, Marks 6

- The design model of JSP application is called MVC model. The MVC stands for Model-View-Controller. The basic idea in MVC design model is to separate out design logic into three parts - modelling, viewing and controlling.
- Any server application is classified in three parts such as business logic, presentation and request processing.
- The **business logic** means the coding logic applied for manipulation of application data. The **presentation** refers to the code written for look and feel of the web page. For example: background color, font style, font size, placing of controls such as text boxes, command buttons and so on. The request processing is nothing but a combination of business logic and presentation. The request processing is always done in order to generate the response.

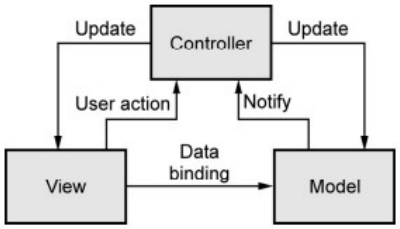


Fig. 4.6.1 Model View Controller (MVC)

- According to the MVC design model, the model corresponds to business logic, view corresponds to presentation and controller corresponds to request processing.

Advantages of using MVC design model

- The use of MVC architecture allows the developer to keep the separation between business logic, presentation and request processing.
- Due to this separation it becomes easy to make **changes** in presentation without disturbing the business logic. The changes in presentation are often required for accommodating the new presentation interfaces.

✅ 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]

=>

Q2. Explain Struts Framework with respect to Architecture, Actions, Interceptors & Exception Handling. [9 Marks]

---

1. Introduction to Struts Framework:

Apache **Struts** is an **open-source web application framework** used for building Java EE web applications. It follows the **Model-View-Controller (MVC)** design pattern to separate concerns, promoting organized and maintainable code.

---

2. Struts Architecture (MVC-based)

The architecture of Struts is based on the MVC pattern:

4.12 Architecture **SPPU : May-18,19, Marks 6**

- The struts framework is based on **M**odel, **V**iew and **C**ontroller architecture (MVC).

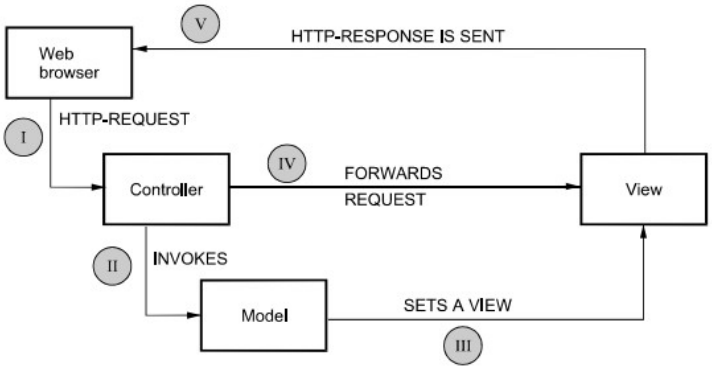
I. Normally when client wants any web page he demands it using get or post request or by clicking Submit button, then a controller is invoked.

II. A Controller is nothing but a Java class in which **business logic** is written. The job of Controller is to take user input and pass it to the **Model**. The controller in turn invokes the **Model.**

III. Model stores the user's data may be in databases or so, and returns some result to the Controller.

IV. Using this result, the Controller figures out user requirements and accordingly forwards the result to the View page.

V. Finally required view can be obtained as a response (i.e. a display of required web page is obtained) on the web browser. This can be diagrammatically shown as below -



In struts,

- **Model** components provide a model for "business logic" and data behind struts program.
- **View** components in struts are those components that present the information to the user or accept the user input. Typically these components are JSP or HTML pages. View components are responsible for providing the display of required information. Struts provide large number of JSP custom tags or **struts tags** to simplify the development of view components.
- **Controllers** are the Java classes which allow developers to remove error handling activities from JSP pages. These classes are typically referred as **action classes**.

i. Model:

- Represents the business logic or data.
- Managed via **JavaBeans, EJBs**, or **POJOs**.
- Handles database access, calculations, and business rules.

ii. View:

- Created using technologies like **JSP**, HTML, or other templating systems.
- Responsible for rendering the user interface.

iii. Controller:

- Central component is the **ActionServlet** (provided by Struts).
- It receives client requests, consults configuration files, and routes the request to the appropriate **Action class**.

---

3. Request Flow in Struts:

1. User submits a request via a web page (JSP).
2. **ActionServlet** receives the request.
3. It consults **struts-config.xml** to identify the appropriate **Action class**.
4. Request is passed through **Interceptors (if using Struts2)**.
5. The specified **Action class** is invoked to handle the request.
6. The Action class returns an **ActionForward** object with a logical name (like "success", "error").
7. The framework maps this logical name to a view (JSP) and sends the response back to the user.

---

4. Actions in Struts:

- An **Action** class in Struts handles user input and contains business logic.
- It extends `org.apache.struts.action.Action`.
- Contains the `execute()` method which:
  - Accepts request/response.
  - Processes the request.
  - Returns an `ActionForward` to indicate the next view.

Example:

```
java
CopyEdit

public class LoginAction extends Action {
  public ActionForward execute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) throws
Exception {

    // Business logic
    return mapping.findForward("success");
  }
}
```

---

5. Interceptors in Struts2:

- **Interceptors** are used to perform **pre-processing and post-processing** of requests.
- Applied before and after the execution of Action methods.
- Common uses:
  - Input validation
  - File upload handling
  - Logging
  - Authentication

Struts2 provides built-in interceptors like:

- `params` (for parameter population)
- `validation` (for form validation)
- `exception` (for exception mapping)

Custom interceptors can also be created by implementing the `Interceptor` interface.

---

6. Exception Handling in Struts:

Struts provides two levels of **exception handling:**

i. Global Exception Handling:

Defined in `struts-config.xml` to handle exceptions across all actions.

```
xml
```

```
CopyEdit

<global-exceptions>
  <exception key="error.login" type="com.example.LoginException"
path="/error.jsp"/>
</global-exceptions>
```

ii. Action-specific Exception Handling:

Configured inside specific action mappings.

```
xml
CopyEdit

<action path="/login" ...>
  <exception key="error.login" type="com.example.LoginException"
path="/loginError.jsp"/>
</action>
```

- When an exception is thrown, Struts checks the mappings and forwards to the appropriate error page.

---

7. Conclusion:

The **Struts framework** simplifies Java web development by enforcing MVC architecture and offering built-in support for **Action classes**, **Interceptors**, and **Exception handling**. Its modular and extensible design enables scalable and maintainable enterprise-level applications.

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

=>

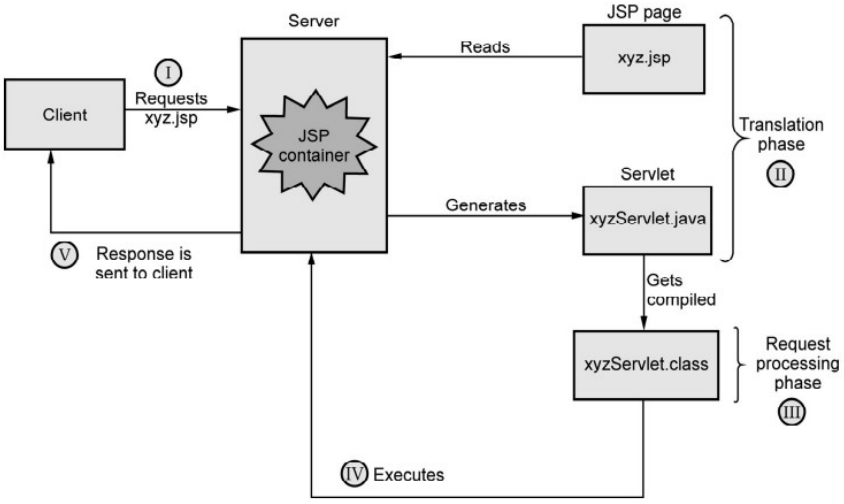


Fig. 4.3.2 JSP processing

1. JSP Lifecycle

JSP (JavaServer Pages) lifecycle defines the steps a JSP page goes through from creation to destruction on the server. It is managed by the **Servlet container** (like Tomcat) and involves **translation, compilation, and execution**.

Phases of JSP Lifecycle:

---

i. Translation Phase:

- The JSP file is **converted into a servlet** by the web container.
- A `.java` file is created with servlet code.

---

ii. Compilation Phase:

- The generated `.java` servlet file is **compiled into a `.class` file** (bytecode).
- This file is loaded into memory.

---

iii. Initialization (`jspInit()`):

- Called only **once** when the JSP is loaded.
- Used to initialize resources like DB connections, config setup.

---

iv. Request Processing (`_jspService()`):

- Called **for every request**.
- Contains the logic written in JSP (HTML + Java).
- Handles request and response objects.

---

v. Destruction (`jspDestroy()`):

- Called **only once** when the JSP is removed from memory or server shuts down.
- Used to release resources like closing DB connections.

JSP Lifecycle Diagram (Optional in exam):

```
scss
CopyEdit
JSP File
   ↓ (Translation)
Servlet Code (.java)
   ↓ (Compilation)
Servlet Class (.class)
   ↓
Initialization → Service → Destruction
```

---

2. Difference Between JSP and Servlet (Min. 4 Points)

Aspect	JSP (JavaServer Pages)	Servlet
1. Syntax Orientation	Tag-based (HTML with Java code)	Pure Java code (programmatic)
2. Purpose	Mainly for **view/presentation layer**	Used for **controller/business logic**
3. Code Complexity	Easier to write and maintain for UI developers	More complex for UI, suitable for logic
4. Modification	Can be modified without recompiling the whole app	Requires recompilation & redeployment
5. Lifecycle Method	`jspInit()`, `_jspService()`, `jspDestroy()`	`init()`, `service()`, `destroy()`
6. Translation	Translated to Servlet by container	Already a compiled Java class

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

=>

1. Concept of Web Services:

- A **Web Service** is a **standardized way** of integrating **web-based applications** using **XML, SOAP, WSDL, and UDDI** over an **HTTP** protocol.
- It enables communication between different software applications running on different platforms and frameworks.
- Web services follow the **Service-Oriented Architecture (SOA)**.

Key Features:

- Platform independent

- Language independent
- Based on open standards (HTTP, XML, SOAP)
- Interoperable across networks and devices

Definition : The Web Services are the software systems that are displayed by the web browser using the web protocol. These software systems are used by the some software applications rather than by end-users directly.

- Web service is a software system designed which is **independent** of specific hardware or software on which it is running.
- **Examples** of a web service are -
  1. **Credit card validation system -** For using this service the client simply enters the ID and password for the credit card and web service reports the validity of it.
  2. **Whether forecast system -** The clients submits the location and the web service returns the information about current whether as well as it may report some predictions.
  3. **Currency converter -** The client enter the source and destination currency along with the amount and appropriate converted information will be displyed.

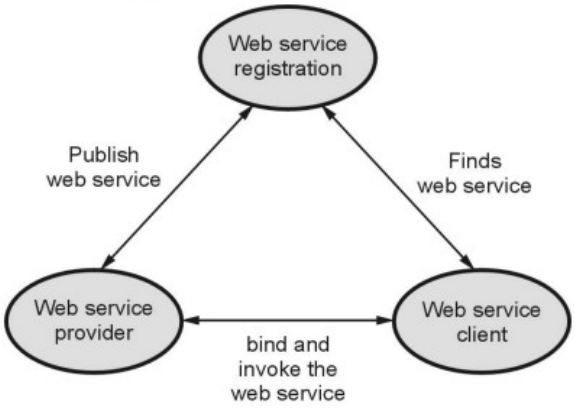


Fig. 4.7.1 Basic web service model

How Web Services Work:

1. A client sends a request to the web service.
2. The service processes the request and sends back the response.
3. Communication is done using **XML-based messages** over protocols like **HTTP**.

---

2. WSDL (Web Services Description Language):

- WSDL is an **XML-based language** used to **describe web services**.

- It defines the **methods, input/output parameters**, data types, **service location**, and **binding protocols**.
- Acts like a **contract** between the client and the server.

Structure of WSDL includes:

- `<types>`: Data types used by the web service.
- `<message>`: Defines input/output messages.
- `<portType>`: Defines web service operations (like functions).
- `<binding>`: Specifies the protocol (e.g., SOAP/HTTP).
- `<service>`: Provides the endpoint URL.

---

3. SOAP (Simple Object Access Protocol):

- SOAP is a **protocol** for exchanging **structured information** in web services.
- It uses **XML** to format messages and is transported via **HTTP or SMTP**.

SOAP Message Structure:

```
xml
CopyEdit

<soap:Envelope>
  <soap:Header>...</soap:Header>
  <soap:Body>
    <m:GetDetails>
      <m:ID>123</m:ID>
    </m:GetDetails>
  </soap:Body>
</soap:Envelope>
```

Key Features of SOAP:

- Platform and language independent
- Extensible and standardized
- Supports RPC (Remote Procedure Call)
- Allows communication over multiple protocols (HTTP, SMTP, etc.)

Conclusion:

Web services provide a universal platform for integrating applications over the internet. **WSDL** describes what the service does, while **SOAP** defines **how** to communicate with the service in a platform-neutral way.

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

=>

1. Concept of JSP (JavaServer Pages):

- **JSP** is a **server-side technology** used to create **dynamic, platform-independent web pages**.
- It allows embedding **Java code into HTML** using special JSP tags.
- JSP is a part of the **Java EE** platform and is executed on the server by the **Servlet container** (e.g., Apache Tomcat).
- It simplifies the creation of web pages by separating **presentation logic** from **business logic**.

---

2. JSP Syntax:

JSP provides special tags to embed Java code into HTML. Key syntax elements:

Element	Syntax	Purpose
Directive	`<%@ %>`	Defines page-level settings
Scriptlet	`<% %>`	Embed Java code
Expression	`<%= %>`	Output result of Java expression
Declaration	`<%! %>`	Declare methods/variables

---

3. Sample JSP Example:

```
jsp
CopyEdit

<%@ page language="java" contentType="text/html" %>
<html>
<head><title>JSP Example</title></head>
<body>
  <h2>Welcome to JSP!</h2>
  <%
    String name = "Student";
    int year = 2025;
  %>
  <p>Hello, <%= name %>!</p>
  <p>Year: <%= year %></p>
</body>
</html>
```

Output:

```
vbnet
CopyEdit

Welcome to JSP!
Hello, Student!
Year: 2025
```

---

4. JSP and Servlet Analogy:

JSP and Servlets are **complementary technologies** in Java web development. JSP is **converted into a Servlet** internally by the server.

Aspect	JSP	Servlet
Focus	View (Presentation layer)	Controller/Business logic
Syntax Style	HTML with embedded Java	Pure Java code
Ease of Design	Easier for designing UI	More complex for UI
Lifecycle	`jspInit()`, `_jspService()`, `jspDestroy()`	`init()`, `service()`, `destroy()`
Conversion	Converted to Servlet at runtime	Already a compiled Java class

---

Analogy (Simplified):

- Think of a **JSP** like a **PowerPoint presentation** with live data updates – designed for **displaying** content.
- A **Servlet** is like the **backend control panel** – handling **logic, inputs, and processing**.
- JSP uses Servlets **internally** but is more **design-friendly**.

Conclusion:

JSP provides a simplified way to create dynamic web pages using Java. It is best suited for presentation, while Servlets are more suitable for controlling application flow and logic. Together, they form the backbone of Java-based web applications.

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

=>

✅ 1. WSDL (Web Services Description Language):

- WSDL is an **XML-based language** used to **describe** the functionalities offered by a **Web Service**.
- It acts like a **contract** between the web service provider and client.
- Defines the **operations**, **messages**, **data types**, **protocols**, and **endpoints** involved in a web service.

◆ Key Elements of WSDL:

Tag	Description
`<types>`	Defines the data types used by the service
`<message>`	Defines input and output messages
`<portType>`	Describes operations provided by the service
`<binding>`	Specifies the communication protocol (like SOAP/HTTP)
`<service>`	Defines the endpoint (URL) of the web service

□ Example (WSDL snippet):

```
xml
CopyEdit

<wsdl:service name="WeatherService">
  <wsdl:port binding="tns:WeatherBinding" name="WeatherPort">
    <soap:address location="http://example.com/weather"/>
  </wsdl:port>
</wsdl:service>
```

✅ 2. SOAP (Simple Object Access Protocol):

- SOAP is a **protocol** used for exchanging **structured information** between systems in **Web Services**.
- It uses **XML format** and typically runs over **HTTP** or **SMTP**.
- SOAP enables **platform-independent communication** between applications.

◆ SOAP Message Structure:

```
xml
CopyEdit

<soap:Envelope>
```

```
  <soap:Header>...</soap:Header>
  <soap:Body>
    <m:GetInfo>
      <m:ID>123</m:ID>
    </m:GetInfo>
  </soap:Body>
</soap:Envelope>
```

◆ Main Components:

Part	Purpose
`Envelope`	Root element of the message
`Header`	Contains metadata (optional)
`Body`	Contains actual message content
`Fault`	Contains error info if any (optional)

✓ WSDL and SOAP Relationship:

- **WSDL** describes *what* the service does and *how* to access it.
- **SOAP** defines *how* to send and receive messages to and from the web service.
- A WSDL document usually includes **SOAP bindings** for communication.

✓ Conclusion:

- **WSDL** provides the **blueprint** for web services, while **SOAP** provides the **means of communication**.
- Together, they enable standardized, interoperable web services in distributed systems.

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

=>

✓ 1. Introduction to MVC Architecture:

MVC (Model-View-Controller) is a **design pattern** that separates an application into **three interconnected components**, improving modularity and ease of maintenance.

4.6 Support for the Model-View-Controller Paradigm

SPPU : Dec.-18, Marks 6

- The design model of JSP application is called MVC model. The MVC stands for Model-View-Controller. The basic idea in MVC design model is to separate out design logic into three parts - modelling, viewing and controlling.
- Any server application is classified in three parts such as business logic, presentation and request processing.
- The **business logic** means the coding logic applied for manipulation of application data. The **presentation** refers to the code written for look and feel of the web page. For example: background color, font style, font size, placing of controls such as text boxes, command buttons and so on. The request processing is nothing but a combination of business logic and presentation. The request processing is always done in order to generate the response.

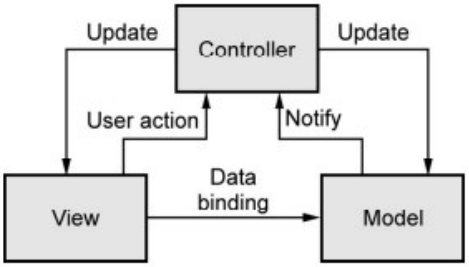


Fig. 4.6.1 Model View Controller (MVC)

- According to the MVC design model, the model corresponds to business logic, view corresponds to presentation and controller corresponds to request processing.

Advantages of using MVC design model

- The use of MVC architecture allows the developer to keep the separation between business logic, presentation and request processing.
- Due to this separation it becomes easy to make **changes** in presentation without disturbing the business logic. The changes in presentation are often required for accommodating the new presentation interfaces.

◆ MVC Components:

Component	Responsibility
Model	Represents business logic and data (Java Beans, POJOs, Database access).
View	Handles the presentation layer (JSP).
Controller	Processes user input and controls the flow (Servlet).

✓ 2. Role of JSP in MVC:

In the MVC model:

- **JSP acts as the View** component.
- It is responsible for displaying the data sent by the controller in a user-friendly format.
- JSP should **not contain business logic**—it only renders the output.

✓ 3. Flow of MVC using JSP:

1. **User Request**: User sends a request via the browser.
2. **Controller (Servlet)**: Receives the request, interacts with the Model to process logic/data.
3. **Model**: Fetches/updates data from the database or business layer.
4. **Controller**: Sets data in `request` or `session` scope and forwards it to the JSP.
5. **JSP (View)**: Extracts data from request and displays it as HTML.

✓ 4. Example: Simple Login Using MVC

◆ Model (LoginBean.java)

```
java
CopyEdit

public class LoginBean {
    private String username, password;
    public boolean validate() {
        return username.equals("admin") && password.equals("admin123");
    }
}
```

◆ Controller (LoginServlet.java)

```
java
CopyEdit

@WebServlet("/login")
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
        String user = req.getParameter("user");
        String pass = req.getParameter("pass");

        LoginBean bean = new LoginBean();
        bean.setUsername(user);
        bean.setPassword(pass);

        if (bean.validate()) {
            req.setAttribute("name", user);
```

```
            req.getRequestDispatcher("welcome.jsp").forward(req, res);
        } else {
            req.getRequestDispatcher("error.jsp").forward(req, res);
        }
    }
}
```

◆ View (welcome.jsp)

```
jsp
CopyEdit

<html>
<body>
  <h2>Welcome, <%= request.getAttribute("name") %>!</h2>
</body>
</html>
```

✓ 5. Advantages of Using JSP with MVC:

- **Separation of Concerns**: UI, logic, and data are separate.
- **Easier Maintenance**: Each part can be modified independently.
- **Reusability**: Components can be reused across multiple modules.
- **Clean Architecture**: Promotes clean, testable code.

✓ Conclusion:

JSP fits perfectly in the **View** part of MVC architecture. It enables developers to **cleanly separate UI** from business logic handled by **Servlets (Controller)** and **Java Beans (Model)**, making web applications more organized, scalable, and maintainable.

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

=>

✓ 1. Introduction to Struts Framework:

- **Struts** is an **open-source web application framework** for developing Java EE web applications using the **Model-View-Controller (MVC)** design pattern.
- Developed by **Apache**, it simplifies the development of large-scale, maintainable web applications.

✅ 2. Struts Architecture:

The **Struts 2** architecture is built on top of **Servlet** and **Filter** technologies and includes the following components:

◆ Main Components:

Component	Role
View (JSP)	Presentation layer that displays data
Controller (FilterDispatcher)	Receives user request and routes it
Model (Java classes/Beans/DB)	Contains business logic and data
Action Classes	Act as a bridge between Model and View
Interceptor Stack	Executes cross-cutting tasks (like logging, validation)
struts.xml	Configuration file for routing and settings

✅ 3. Struts Workflow:

1. **User Request**: User sends a request (e.g., `login.action`).
2. **FilterDispatcher**: Receives the request and looks for the mapping in `struts.xml`.
3. **Interceptor Stack**: Pre-processes request (validation, authentication, etc.).
4. **Action Class**: Executes business logic and returns a result.
5. **Result (JSP)**: Based on the result, appropriate JSP page is rendered.
6. **Response**: Response is sent back to the user.

✅ 4. Action in Struts:

- An **Action class** processes the user request and interacts with the model.
- It returns a **String result name** (like “success”, “error”) mapped in `struts.xml`.

🔹 Example:

```
java
CopyEdit

public class LoginAction extends ActionSupport {
    private String username, password;
```

```
    public String execute() {
        if (username.equals("admin") && password.equals("admin123"))
            return "success";
        else
            return "error";
    }

    // Getters and Setters
}
```

✅ 5. Interceptors in Struts:

- Interceptors are used to **process requests and responses** before and after the action executes.
- Examples: **ValidationInterceptor, LoggingInterceptor, TimerInterceptor**.

◆ Key Uses:

- Input validation
- Pre-processing (e.g., trimming, token verification)
- Post-processing (e.g., logging)

✅ 6. Exception Handling in Struts:

- Struts allows **centralized exception handling** via `struts.xml`.
- Developers can configure global or action-specific exception mappings.

□ Example in `struts.xml`:

```
xml
CopyEdit

<global-exceptions>
    <exception key="error.login" type="java.lang.Exception"
path="/error.jsp"/>
</global-exceptions>
```

This ensures the user is redirected to a proper error page when an exception occurs.

✅ Conclusion:

Struts framework follows the **MVC design pattern** and supports **clean separation of concerns**. With the help of **Action classes**, **Interceptors**, and **centralized exception handling**, Struts enables developers to build robust and scalable enterprise-level web applications.

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

=>

✓ Introduction to Interceptors in Struts:

- **Interceptors** are a core feature of the **Struts 2 framework**.
- They are used to **intercept requests and responses** before or after the execution of an **Action**.
- They perform **cross-cutting tasks** like logging, validation, file upload, etc.

Struts 2 comes with a rich set of **built-in interceptors** that enhance application capabilities.

✓ Important Interceptors in Struts Framework:

Interceptor	Description
params interceptor	Automatically maps request parameters to Action class properties (default behavior).
validation interceptor	Performs server-side validation before executing the Action. Uses `validate()` method or `validation.xml`.
workflow interceptor	Works with `validation` interceptor. If validation fails, it prevents Action execution and returns to input page.
modelDriven interceptor	Used when the Action class implements `ModelDriven` interface; makes a model object accessible to the framework.
fileUpload interceptor	Handles file upload from forms; parses `multipart/form-data` and stores uploaded files.
prepare interceptor	Calls the `prepare()` method of `Preparable` interface before `execute()`; used for loading data before action.
exception interceptor	Catches exceptions thrown by the Action and allows custom exception handling (e.g., redirecting to an error page).
logger interceptor	Logs the request and response parameters and actions for debugging and tracking.

✓ Conclusion:

Struts interceptors are **powerful tools** for managing common concerns across multiple actions in a **modular and reusable** way. By using built-in or custom interceptors, developers can simplify their code and enforce consistent behavior throughout the application.

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

=>

✓ Introduction:

Web Services are platform-independent, standardized ways to allow communication and data exchange between different software applications over the **Internet** using **XML**, **SOAP**, **WSDL**, or **REST**.

They enable **interoperability** between different systems and are widely used in modern distributed systems.

✓ Benefits of Using Web Services:

Benefit	Justification
1. Platform & Language Independence	Web services use standard protocols (HTTP, XML, JSON), so they can work across different platforms (Windows, Linux) and languages (Java, .NET, PHP).
2. Interoperability	Web services enable communication between heterogeneous systems (e.g., Java app calling .NET service).
3. Reusability	A single web service can be reused by multiple applications (e.g., a currency converter service used by multiple banks).
4. Loose Coupling	Web services allow clients and servers to evolve independently as long as the contract (WSDL/API) remains unchanged.
5. Easy Integration	Web services make it simple to integrate new functionality into existing applications without redesigning the whole system.

✓ Conclusion:

Web Services offer a **standard, scalable, and flexible solution** for integrating applications across platforms and networks. Their ability to promote **reuse, interoperability, and platform neutrality** makes them a key component in enterprise and web-based application development.

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

=>

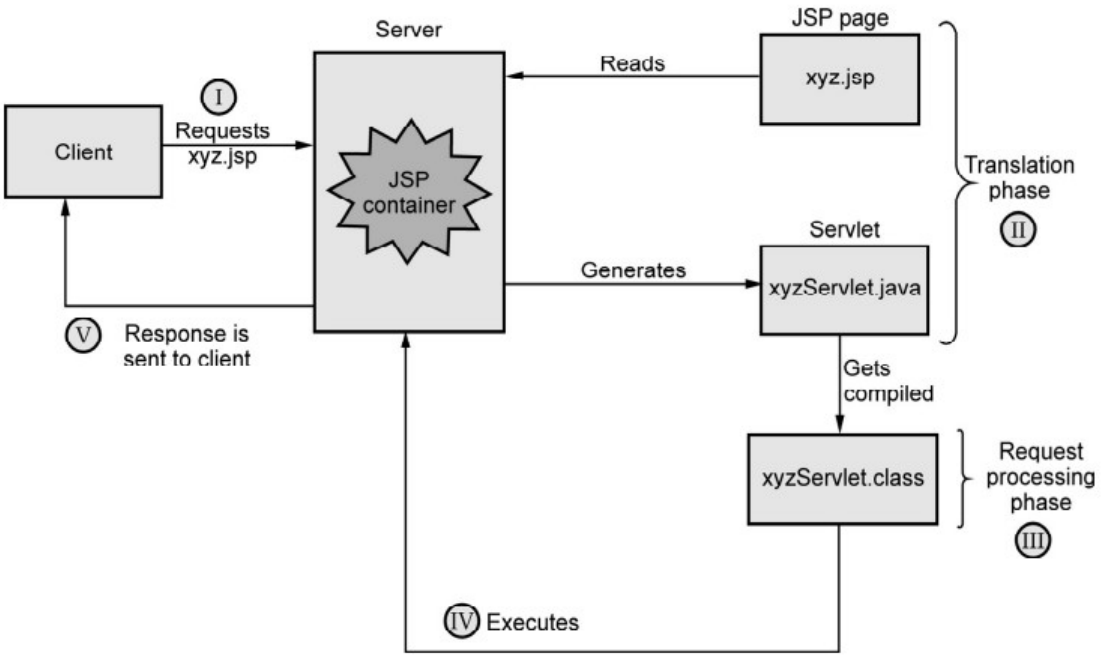


Fig. 4.3.2 JSP processing

✓ Introduction:

The **JSP (JavaServer Pages) lifecycle** defines the **phases** a JSP page goes through, from **translation** to **destruction**. It is handled by the **JSP container** (like Apache Tomcat), which converts JSP into a servlet and manages its execution.

✓ JSP Life Cycle Phases:

◆ 1. Translation Phase

- The JSP page is **converted into a servlet** by the container.
- Example: `login.jsp` → `login_jsp.java`

◆ 2. Compilation Phase

- The generated Java file is **compiled** into a `.class` file.
- Example: `login_jsp.java` → `login_jsp.class`

◆ 3. Loading and Instantiation

- The **compiled servlet class** is **loaded** into memory and an **instance** is created.

◆ 4. Initialization (`jspInit()`)

- Called **once** in the lifecycle.
- Used to initialize resources like database connections.

◆ 5. Request Processing (`_jspService()`)

- Called **every time** the JSP page is requested.
- Processes client requests and generates dynamic response.

◆ 6. Destruction (`jspDestroy()`)

- Called when the JSP is taken out of service.
- Used to release resources (e.g., closing database connections).

✓ JSP Life Cycle Diagram:

```
scss
CopyEdit
   Client Request
         ↓
  [JSP File: index.jsp]
         ↓
Translation Phase
(JSP → Servlet)
         ↓
Compilation Phase
(.java → .class)
         ↓
Class Loading & Object Creation
         ↓
Initialization (jspInit())
         ↓
Request Handling (_jspService())
         ↓
Response Sent to Client
         ↓
Destruction (jspDestroy())
```

(Draw this as a labeled flowchart in your answer sheet)

✅ Conclusion:

The JSP lifecycle is essential for understanding how JSP pages are **executed on the server**. Key methods like `jspInit()`, `_jspService()`, and `jspDestroy()` ensure proper initialization, request handling, and cleanup.

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

=>

What is JSP?

- **JSP (JavaServer Pages)** is a **server-side technology** that helps create **dynamic web pages**.
- It allows embedding **Java code** directly inside **HTML** using special tags.
- JSP is compiled into a **Servlet** by the server at runtime.

Advantages of JSP over Servlet:

Advantage	Explanation
1. Easier to write and maintain	JSP allows mixing HTML and Java code naturally, unlike servlets where HTML is embedded in Java code.
2. Separation of presentation	JSP focuses on presentation (view), making it easier to separate UI from business logic.
3. Built-in objects support	JSP provides implicit objects like `request`, `response`, `session`, simplifying development.
4. Better for designers	Designers can edit JSP files as HTML pages without deep Java knowledge.
5. Automatic compilation	JSP pages are compiled automatically into servlets by the server, reducing developer effort.

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

=>

✓ Introduction:

- **MVC (Model-View-Controller)** is a **software design pattern** used to develop web applications.
- It **separates** the application into **three interconnected components**:
  - **Model** — handles data and business logic
  - **View** — presents data to the user (UI)
  - **Controller** — processes user requests and controls the flow

---

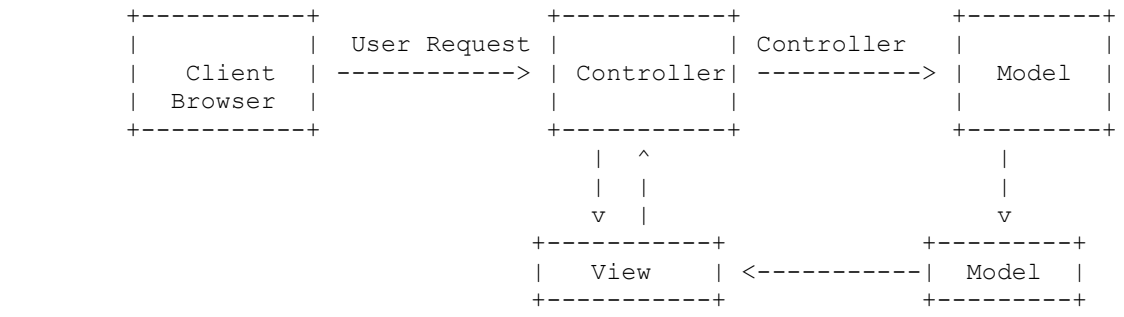
✓ MVC Components:

Component	Role
Model	Manages data, database operations, and business logic. Notifies the View when data changes.
View	Displays the data to the user. Generates UI based on Model data. Examples: JSP, HTML pages.
Controller	Handles user input and interactions, processes requests, updates the Model, and selects the View to display.

✓ Working of MVC Architecture:

1. **User sends a request** via browser (click, form submit).
2. **Controller receives the request**, processes it (e.g., validation).
3. Controller interacts with the **Model** to retrieve or update data.
4. Model performs business logic and updates data.
5. Controller selects the appropriate **View** to render the response.
6. View generates the UI and sends it back to the user's browser.

✓ MVC Architecture Diagram:



✓ Conclusion:

MVC architecture promotes **separation of concerns**, making applications easier to develop, test, and maintain. It improves **scalability** and **code reuse** in web applications.

4.6 Support for the Model-View-Controller Paradigm

SPPU : Dec.-18, Marks 6

- The design model of JSP application is called MVC model. The MVC stands for Model-View-Controller. The basic idea in MVC design model is to separate out design logic into three parts - modelling, viewing and controlling.
- Any server application is classified in three parts such as business logic, presentation and request processing.
- The **business logic** means the coding logic applied for manipulation of application data. The **presentation** refers to the code written for look and feel of the web page. For example: background color, font style, font size, placing of controls such as text boxes, command buttons and so on. The request processing is nothing but a combination of business logic and presentation. The request processing is always done in order to generate the response.

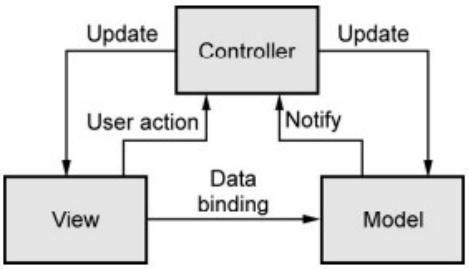


Fig. 4.6.1 Model View Controller (MVC)

- According to the MVC design model, the model corresponds to business logic, view corresponds to presentation and controller corresponds to request processing.

Advantages of using MVC design model

- The use of MVC architecture allows the developer to keep the separation between business logic, presentation and request processing.
- Due to this separation it becomes easy to make **changes** in presentation without disturbing the business logic. The changes in presentation are often required for accommodating the new presentation interfaces.

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

=>

✓ Introduction to Struts Framework

- **Apache Struts** is a popular **MVC-based Java web application framework**.
- It simplifies web app development by separating **Model, View, and Controller**.
- Struts uses **Action classes**, **Form beans**, and **XML configuration** to manage the application flow.

✓ Struts Architecture Components

Component	Description
Client	Sends HTTP request from browser.
Controller (ActionServlet)	Central servlet that receives requests and manages application flow based on configuration.
ActionMapping	Reads configuration and decides which Action class to call.
ActionForm	Java bean to hold form data submitted by client.
Action	Processes the business logic and interacts with the Model (e.g., database).
Model	Represents the business data and logic (usually plain Java classes or EJBs).
View (JSP)	Generates the HTML response to the client.

✓ Working Flow of Struts Architecture:

1. The **client sends a request** (e.g., form submission).
2. The **ActionServlet** receives the request.
3. ActionServlet consults the **struts-config.xml** to find the appropriate **ActionMapping**.
4. ActionServlet creates or retrieves an **ActionForm** bean and populates it with request parameters.
5. ActionServlet calls the **validate()** method on the form bean (if any validation logic exists).
6. After validation, ActionServlet calls the corresponding **Action class's execute()** method.
7. The **Action class** processes business logic and returns an **ActionForward** object specifying the next view.
8. The **controller forwards** the request to the specified **JSP page** (View).
9. The JSP generates the response to be sent back to the client.

4.12 Architecture

SPPU : May-18,19, Marks 6

- The struts framework is based on **M**odel, **V**iew and **C**ontroller architecture (MVC).

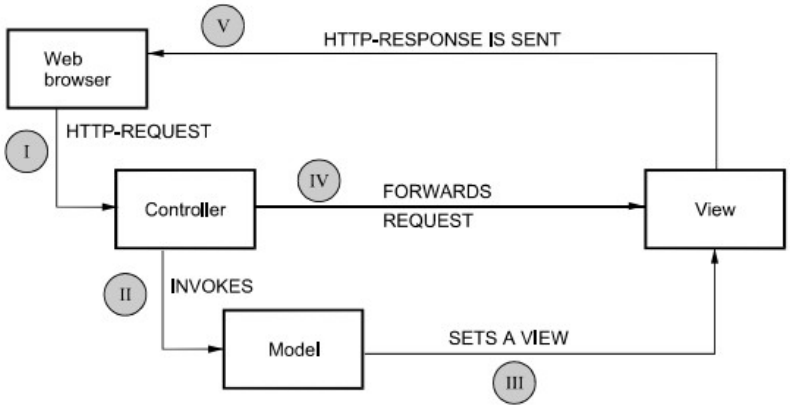
I. Normally when client wants any web page he demands it using get or post request or by clicking Submit button, then a controller is invoked.

II. A Controller is nothing but a Java class in which **business logic** is written. The job of Controller is to take user input and pass it to the **Model**. The controller in turn invokes the **Model.**

III. Model stores the user's data may be in databases or so, and returns some result to the Controller.

IV. Using this result, the Controller figures out user requirements and accordingly forwards the result to the View page.

V. Finally required view can be obtained as a response (i.e. a display of required web page is obtained) on the web browser. This can be diagrammatically shown as below -



✓ Struts Architecture Diagram:

```
pgsql
CopyEdit
  Client Request
       ↓
+------------------+
|  ActionServlet   |  <--- Controller
+------------------+
       ↓
+------------------+
|  struts-config.xml|  <--- Configuration file
+------------------+
       ↓
+------------------+       +-------------+
|   ActionForm     |       |   Action    |
|  (Form Bean)     |       | (Business)  |
+------------------+       +-------------+
       ↓                          ↓
       +-----------→ Model (Business Logic & Data)
                                  ↓
                            ActionForward
```

```
        ↓
View (JSP)
        ↓
Response to Client
```

✅ Benefits of Struts Framework:

Benefit	Explanation
1. MVC Design Pattern	Clear separation of Model, View, and Controller improves application structure and maintainability.
2. Reusability	Components like Action classes and Form beans can be reused across the application.
3. Centralized Configuration	Using `struts-config.xml` allows easy management of all mappings and resources in one place.
4. Built-in Tag Libraries	Struts provides custom JSP tags simplifying UI development and reducing Java code in JSPs.
5. Validation Framework	Supports client-side and server-side validation easily with configurable rules.
6. Internationalization	Supports multiple languages and locales for global applications.
7. Extensibility	Highly extensible and supports integration with other Java EE technologies.

✅ Conclusion:

Struts framework, through its MVC architecture, helps in building scalable, maintainable, and well-structured Java web applications. Its clear separation of concerns and rich features like validation, internationalization, and tag libraries simplify development and improve productivity.

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

=>

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

```
jsp
CopyEdit

<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
<%-- JSP Comment: This is a JSP comment and will NOT appear in the HTML
output --%>

<!DOCTYPE html>
<html>
<head>
    <title>JSP Directive, Scriptlet, Expression & Comment Demo</title>
</head>
<body>

<%
    // Scriptlet: Java code inside JSP
    String name = "ChatGPT";
    int year = 2025;
%>

<h1>Welcome to <%= name %> JSP Tutorial</h1>  <%-- Expression: Outputs
value to HTML --%>
<p>Current Year: <%= year %></p>

<%-- Another JSP Comment --%>
<p>This demonstrates usage of:</p>
<ul>
    <li>Page Directive</li>
    <li>Scriptlet</li>
    <li>Expression</li>
    <li>JSP Comment</li>
</ul>

</body>
</html>
```

Explanation:

Feature	Explanation
Page Directive	`<%@ page ... %>` sets JSP page settings like language, content type, and encoding.
Scriptlet	`<% ... %>` allows embedding Java code within JSP.
Expression	`<%= ... %>` evaluates and inserts the result directly into the HTML output.

Feature	Explanation
JSP Comment	`<%-- ... --%>` is a comment in JSP, not sent to client browser (unlike HTML comments).

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

=>

Benefits of Web Services: (4 Marks)

Benefit	Explanation
1. Interoperability	Web services allow communication between different applications built on different platforms and languages.
2. Reusability	Services can be reused by multiple applications, reducing redundancy and development effort.
3. Loose Coupling	Web services are loosely coupled, meaning client and server can evolve independently.
4. Standardized Protocols	Use of standards like HTTP, XML, SOAP, REST ensures compatibility and easy integration.
5. Accessibility	Accessible over the internet or intranet via standard web protocols.
6. Ease of Integration	Simplifies integration of heterogeneous systems in distributed environments.

Explanation of SOAP, REST, and UDDI: (4 Marks)

1. SOAP (Simple Object Access Protocol):

- A **protocol** for exchanging structured information in web services using **XML**.
- Works over HTTP, SMTP, or other protocols.
- Uses **XML messages** for request and response.
- Supports **RPC (Remote Procedure Call)** style messaging.
- Highly standardized with strict rules and built-in error handling.
- Enables **formal contracts** using WSDL (Web Services Description Language).

---

2. REST (Representational State Transfer):

- An **architectural style** for designing networked applications.
- Uses **standard HTTP methods**: GET, POST, PUT, DELETE.
- Uses URLs to represent resources and returns data typically in **JSON** or **XML**.
- Stateless communication, meaning each request contains all info to process it.
- Lightweight, faster, and easier to use compared to SOAP.
- Widely used for web APIs and modern web applications.

---

3. UDDI (Universal Description, Discovery, and Integration):

- A **platform-independent, XML-based registry** for listing and discovering web services.
- Acts like a **directory** for businesses to publish their web services.
- Allows service consumers to **find** appropriate web services.
- Supports **service metadata** for categorization and description.
- Enables dynamic binding between service providers and consumers.

Summary Table:

Term	Description	Communication Style	Data Format
SOAP	Protocol for exchanging XML messages	Strict, XML-based, RPC style	XML
REST	Architectural style for APIs	HTTP verbs, resource-oriented	JSON, XML, HTML etc.
UDDI	Registry for discovering web services	Directory service	XML

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

=>

✓ 1. Introduction to Struts Framework:

- **Apache Struts** is an **open-source MVC framework** used to build **Java EE web applications**.
- It follows the **Model-View-Controller (MVC)** design pattern, helping to separate business logic, user interface, and request handling.

- Struts simplifies the development and maintenance of large-scale web applications.

✓ 2. Struts Framework Components: (5 Marks)

Component	Description
1. View (JSP/HTML)	The user interface part. JSP files are used to display data to the user and collect input.
2. Controller (ActionServlet)	Receives all client requests. It acts as the controller in MVC, routing requests to appropriate handlers.
3. Model (JavaBeans/Database/Services)	Represents business logic and data layer, such as Java classes handling data operations.
4. ActionForm (Form Bean)	A JavaBean that holds form data from user input. It is automatically populated by the controller.
5. Action Class	Processes the request, interacts with the model, and returns a result.
6. struts-config.xml	Configuration file to map requests to actions and views. Central to the application flow.

✓ Flow of Request in Struts:

1. User submits a form (e.g., login.jsp).
2. The **ActionServlet** receives the request.
3. It looks up `struts-config.xml` to determine the mapping.
4. Form data is populated into **ActionForm**.
5. The **Action class** executes business logic.
6. An **ActionForward** determines which view (JSP) to display.
7. The view is returned to the user.

✓ 3. Interceptors in Struts 2: (4 Marks)

- **Interceptors** are key components in **Struts 2**, used to **pre-process and post-process** requests and responses.
- They sit between the request and the action execution.
- Interceptors can:

- Perform input validation
- Log request/response data
- Handle exceptions
- Manage file uploads, session, etc.

✅ Common Interceptors in Struts:

Interceptor	Description
params	Sets request parameters on the action class.
validation	Automatically performs input validation using XML or annotations.
workflow	Prevents execution if validation fails.
exception	Maps exceptions to error views.
fileUpload	Handles file uploads from forms.
logger	Logs request data for debugging.

✅ Benefits of Using Interceptors:

- **Modular code**: Pre/post-processing logic can be separated from core action logic.
- **Reusability**: Interceptors can be reused across multiple actions.
- **Custom logic**: Developers can create **custom interceptors** to handle tasks like authentication, logging, etc.

✅ Conclusion:

Struts provides a robust MVC framework for Java web applications. Its component-based structure makes it scalable and maintainable. **Interceptors** in Struts 2 enhance request processing by allowing centralized and reusable logic before and after action execution.

UNIT 5

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

=>

1. Indexed Array (Numerical Array):

- Stores elements with **numeric indexes** starting from 0.
- Values are accessed using these numeric indices.

✅ Syntax:

```
php
CopyEdit

$fruits = array("Apple", "Banana", "Mango");
echo $fruits[1];  // Output: Banana
```

✅ Alternate Syntax:

```
php
CopyEdit

$colors[] = "Red";
$colors[] = "Green";
$colors[] = "Blue";
```

---

2. Associative Array:

- Uses **named keys** (strings) instead of numeric indexes.
- Keys are mapped to specific values like a dictionary.

✅ Syntax:

```
php
CopyEdit

$student = array(
    "name" => "John",
    "age" => 20,
    "course" => "B.Tech"
);

echo $student["course"];  // Output: B.Tech
```

✅ Alternate Syntax:

```
php
CopyEdit

$marks["Math"] = 85;
$marks["Science"] = 90;
$marks["English"] = 88;
```

---

3. Multidimensional Array:

- An array containing **one or more arrays** as elements.
- Used to represent matrices or complex data structures.

✅ Syntax:

```
php
CopyEdit

$students = array(
    array("John", 20, "B.Tech"),
    array("Alice", 21, "BCA"),
    array("Bob", 22, "MBA")
);

echo $students[1][0];  // Output: Alice
echo $students[2][2];  // Output: MBA
```

✅ Summary Table:

Type	Key Type	Access Example
Indexed Array	Numeric	`$array[0]`
Associative Array	String (named)	`$array["key"]`
Multidimensional	Mixed	`$array[i][j]`

✅ Conclusion:

PHP arrays are powerful tools that can store and organize data efficiently. Understanding the three types—**indexed, associative, and multidimensional**—helps developers build flexible and scalable PHP applications.

Q2. Explain the following:

i) WAP & WML [4]

ii) C# Vs Java [5]

=>

i) WAP & WML [4 Marks]

WAP (Wireless Application Protocol):

- WAP is a technical standard developed to allow mobile devices such as cell phones and PDAs to access internet content and services.
- It was designed to overcome the limitations of mobile devices such as small screens, limited bandwidth, and low memory.
- WAP enables the creation of web-like content optimized for wireless networks and devices.
- It supports a range of wireless network types including GSM, CDMA, and GPRS.

Key Features of WAP:

- Lightweight and optimized for mobile.
- Operates using a microbrowser.
- Works over all major wireless networks.

WML (Wireless Markup Language):

- WML is a markup language used to design pages that can be displayed on WAP-enabled mobile devices.
- It is similar to HTML but is specifically designed for small screens and limited device capabilities.
- WML is based on XML and supports navigation between small chunks of content called **"cards"** grouped into **"decks."**

Key Features of WML:

- XML-based language for WAP.
- Supports user interaction via soft keys.
- Provides text, image display, input forms, and navigation.

Conclusion:
WAP is the protocol that enables mobile communication, while WML is the language used to design the mobile-friendly pages delivered via WAP.

---

ii) C# Vs Java [5 Marks]

Feature	C#	Java
Platform	Developed by Microsoft, mainly used in Windows (via .NET Framework / .NET Core)	Developed by Sun Microsystems (now Oracle), platform-independent (Write Once, Run Anywhere using JVM)
Syntax	Similar to Java but also influenced by C++	Similar to C++, simpler than C# in many areas
Compilation	Compiles to Intermediate Language (IL) and runs on CLR (Common Language Runtime)	Compiles to bytecode and runs on JVM (Java Virtual Machine)

Feature	C#	Java
Memory Management	Automatic garbage collection using .NET GC	Automatic garbage collection using JVM GC
Language Features	Supports properties, indexers, delegates, events, LINQ	Rich standard libraries, supports RMI, multithreading, but lacks properties and indexers
Performance	Fast, especially on Windows; optimized with .NET Core	Slightly lower on Windows; highly portable
Development Tools	Visual Studio (feature-rich IDE)	Eclipse, IntelliJ IDEA, NetBeans

Conclusion:
Both C# and Java are object-oriented, secure, and robust languages. Java is known for platform independence and portability, whereas C# offers tight integration with Windows and rapid application development using .NET.

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

=>

Q3. Explain PHP with MySQL using example. [6 Marks]

PHP with MySQL is a common combination used to build dynamic and data-driven web applications.

- **PHP (Hypertext Preprocessor):** A server-side scripting language used to create dynamic web pages.
- **MySQL:** An open-source relational database management system used to store and manage data.

Steps to Use PHP with MySQL:

1. **Connect to MySQL database using PHP**
2. **Perform database operations (INSERT, SELECT, UPDATE, DELETE)**
3. **Display the result or status**

Example: PHP with MySQL

1. Create a MySQL Table:

```
sql
CopyEdit
CREATE DATABASE mydb;
```

```
USE mydb;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50),
    email VARCHAR(100)
);
```

---

2. PHP Script to Insert and Display Data:

```
php
CopyEdit
<?php
// Step 1: Connect to MySQL
$conn = mysqli_connect("localhost", "root", "", "mydb");

if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Step 2: Insert Data into Table
$sql = "INSERT INTO users (name, email) VALUES ('John Doe',
'john@example.com')";
if (mysqli_query($conn, $sql)) {
    echo "New record inserted successfully.<br>";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

// Step 3: Retrieve and Display Data
$result = mysqli_query($conn, "SELECT * FROM users");

echo "<h3>User List:</h3>";
while ($row = mysqli_fetch_assoc($result)) {
    echo "ID: " . $row["id"] . " | Name: " . $row["name"] . " | Email: " .
$row["email"] . "<br>";
}

// Step 4: Close connection
mysqli_close($conn);
?>
```

Explanation:

- `mysqli_connect()` – connects PHP to MySQL.
- `mysqli_query()` – runs SQL commands like INSERT and SELECT.
- `mysqli_fetch_assoc()` – fetches each row as an associative array.
- `mysqli_close()` – closes the database connection.

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

=>

i) Session Tracking in PHP [4 Marks]

Session tracking refers to the technique used to maintain user-specific data across multiple web pages during a user's visit to a website. Since HTTP is stateless (it doesn't remember previous user interactions), session tracking is essential for storing temporary data.

How Session Tracking Works in PHP:

- When a session starts, PHP creates a unique **session ID** for the user.
- This ID is usually stored in a **cookie** called `PHPSESSID`.
- The server creates a file on the server (usually in `/tmp`) to store session variables.
- As the user navigates through different pages, PHP uses the session ID to retrieve the stored data.

PHP Session Functions:

Function	Description
`session_start()`	Initializes a session. Must be called before any output.
`$_SESSION['key']`	Used to store or access session data.
`session_destroy()`	Ends the session and deletes session data.

Example:

store.php

```
php
CopyEdit
<?php
session_start();
$_SESSION['user'] = "John";
echo "Session data is saved.";
?>
```

retrieve.php

```
php
CopyEdit
<?php
session_start();
echo "Welcome, " . $_SESSION['user'];
```

```
?>
```

Use Cases:

- User login systems
- Shopping carts in e-commerce
- User preferences (language, theme, etc.)
- Form data preservation across steps

Advantages:

- Secure since data is stored server-side.
- Reduces the need for repeated database access.
- Sessions expire automatically or can be manually destroyed.

---

ii) .NET Framework [4 Marks]

The **.NET Framework** is a comprehensive software development platform developed by **Microsoft**. It supports the creation and execution of applications on **Windows OS**, including web apps, desktop apps, and services.

Core Components:

1. **CLR (Common Language Runtime):**
   - Executes code and provides services like memory management, type safety, exception handling, etc.
   - Acts like the Java Virtual Machine (JVM) in Java.
2. **FCL (Framework Class Library):**
   - Contains thousands of reusable classes and functions.
   - Supports operations like file access, data collections, database operations, web development, etc.
3. **Languages Supported:**
   - C#, VB.NET, F#, etc.
   - All .NET languages compile to a common intermediate language (CIL).

ASP.NET:

- A part of the .NET Framework used for building web-based applications.
- Supports Web Forms, MVC (Model-View-Controller), and Web APIs.

Features of .NET Framework:

- **Language interoperability** – Multiple languages work together.
- **Memory Management** – Automatic garbage collection.
- **Robust Security Model** – Code access security and role-based security.
- **Easy Deployment** – Uses tools like ClickOnce, XCOPY.
- **Integrated IDE** – Visual Studio provides advanced features.

Real-life Applications:

- Banking portals
- Enterprise business apps
- Windows desktop software
- Government management systems

---

iii) Node.js [4 Marks]

Node.js is a powerful JavaScript runtime built on **Chrome's V8 engine**. It enables developers to use JavaScript for writing **server-side code**, allowing full-stack development using a single language.

Key Concepts:

- **Event-driven architecture:** Executes operations based on events, reducing wait time.
- **Non-blocking I/O:** Handles multiple requests concurrently using callbacks, promises, or async-await.
- **Single-threaded model:** Efficient for I/O-heavy tasks, unlike traditional multi-threaded servers.

Features of Node.js:

- Fast execution due to the V8 engine.
- Built-in libraries for HTTP, file system, stream, and more.
- Open-source with a huge ecosystem via **npm** (Node Package Manager).
- Ideal for microservices architecture.

Code Example:

```
javascript
CopyEdit
const http = require('http');
http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello from Node.js!');
}).listen(3000);
```

Popular Node.js Frameworks:

- **Express.js** – Lightweight web application framework.
- **Socket.io** – For real-time communication (chat apps).
- **NestJS** – Enterprise-level framework for building scalable apps.

Use Cases:

- Real-time applications (chat, gaming)
- RESTful APIs and backend services
- IoT solutions
- Streaming and file upload systems

Advantages of Node.js:

- Uses JavaScript for both frontend and backend.
- High scalability and concurrency.
- Suitable for agile and rapid development.

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

=>

i) ASP.NET [4 to 5 Marks]

ASP.NET stands for **Active Server Pages .NET**. It is a **server-side web development framework** developed by **Microsoft** for building **dynamic web applications, websites, and web services** using .NET technologies.

Key Features of ASP.NET:

1. **Compiled Code:**
    - ASP.NET code is compiled, which improves performance over traditional interpreted scripts like classic ASP or PHP.
2. **Multi-language Support:**
    - You can write ASP.NET code in **C#, VB.NET**, or other .NET-supported languages.
3. **Rich Toolbox in Visual Studio:**
    - Drag-and-drop controls, automatic deployment, debugging tools, etc.
4. **State Management:**
    - Supports various methods like ViewState, Session State, Application State.
5. **Security Features:**
    - In-built Windows authentication, form authentication, role management, and request validation.

ASP.NET Frameworks:

1. **Web Forms:**
    - Simplifies UI development with event-driven programming (like Windows Forms).
    - Uses controls like TextBox, Button, GridView.
2. **MVC (Model-View-Controller):**
    - Promotes separation of concerns.
    - Suitable for large, scalable enterprise apps.
3. **Web API:**
    - Used to build RESTful HTTP services.

Architecture Overview:

```
css
CopyEdit
[ Client (Browser) ]
        ↓
[ ASP.NET Web App ]
        ↓
[ .NET Framework (CLR + FCL) ]
        ↓
[ Operating System (Windows) ]
```

Use Cases:

- Enterprise portals
- Online shopping sites
- Data-driven dashboards
- APIs for mobile apps

Advantages:

- High performance due to compiled code
- Scalable and secure
- Excellent IDE support via **Visual Studio**
- Strong community and Microsoft support

---

ii) Node.js [4 to 5 Marks]

Node.js is an **open-source, cross-platform** JavaScript runtime environment used for developing fast and scalable **server-side** and **network applications**.

Key Characteristics of Node.js:

1. **Event-Driven, Non-blocking I/O:**
   - Efficiently handles thousands of simultaneous connections without threads.
2. **Built on V8 JavaScript Engine:**
   - Fast code execution (same engine used in Google Chrome).
3. **Single-threaded but Highly Scalable:**
   - Uses event loop instead of traditional multi-threading.
4. **npm (Node Package Manager):**
   - Offers over 2 million packages for rapid development.

Core Modules:

- `http` – Create server and handle requests
- `fs` – File system operations
- `path`, `url`, `events` – Useful utility modules

Simple Example:

```
javascript
CopyEdit
const http = require('http');

http.createServer((req, res) => {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Welcome to Node.js server!');
```

```
}).listen(3000);
```

Use Cases:

- Real-time applications (e.g., chat apps)
- REST APIs and backend services
- Streaming and media apps
- IoT and microservices

Advantages of Node.js:

- Full-stack JavaScript (frontend + backend)
- Extremely fast and lightweight
- Ideal for modern, scalable, real-time web applications
- Huge developer community and libraries

Comparison with ASP.NET (optional for extra 0.5 marks):

Feature	ASP.NET	Node.js
Language	C#, VB.NET	JavaScript
Platform	Windows-centric (mostly)	Cross-platform
Architecture	Multi-threaded	Single-threaded, event-based
Best for	Enterprise, portal apps	Real-time, fast APIs

Conclusion:

- **ASP.NET** is a powerful, enterprise-ready framework for developing dynamic web apps using the .NET environment.
- **Node.js** offers fast, event-driven performance for real-time, high-concurrency applications using JavaScript on the server side.

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

=>

Step-by-Step Explanation:

This PHP code connects to a **MySQL database** and retrieves records from a table using **MySQLi (MySQL Improved)** extension.

✅ 1. Create Database and Table (Optional – For Reference)

```
sql
CopyEdit
CREATE DATABASE mydb;
USE mydb;

CREATE TABLE students (
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50),
    email VARCHAR(100)
);

INSERT INTO students (name, email) VALUES
('John Doe', 'john@example.com'),
('Jane Smith', 'jane@example.com');
```

✅ 2. PHP Code to Connect and Display Records

```
php
CopyEdit
<?php
// Step 1: Create connection
$host = "localhost";
$username = "root";
$password = "";
$database = "mydb";

$conn = mysqli_connect($host, $username, $password, $database);

// Step 2: Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Step 3: Query to fetch data
$sql = "SELECT * FROM students";
$result = mysqli_query($conn, $sql);

// Step 4: Display data
if (mysqli_num_rows($result) > 0) {
    echo "<h3>Student Records:</h3>";
    echo "<table border='1' cellpadding='5'>";
    echo "<tr><th>ID</th><th>Name</th><th>Email</th></tr>";
    while ($row = mysqli_fetch_assoc($result)) {
        echo "<tr>";
        echo "<td>" . $row["id"] . "</td>";
        echo "<td>" . $row["name"] . "</td>";
        echo "<td>" . $row["email"] . "</td>";
```

```
        echo "</tr>";
    }
    echo "</table>";
} else {
    echo "No records found.";
}

// Step 5: Close connection
mysqli_close($conn);
?>
```

✅ Output Format:

The output will display the student table like this:

```
sql
CopyEdit
Student Records:

+----+------------+--------------------+
| ID | Name       | Email              |
+----+------------+--------------------+
| 1  | John Doe   | john@example.com   |
| 2  | Jane Smith | jane@example.com   |
+----+------------+--------------------+
```

✅ Explanation:

- **mysqli_connect()**: Connects to the database.
- **mysqli_query()**: Executes the SQL SELECT query.
- **mysqli_fetch_assoc()**: Fetches each row as an associative array.
- **mysqli_close()**: Closes the connection.

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

=>

✅ 1. WAP (Wireless Application Protocol)

[3 Marks – Concept, Features, Architecture, Uses]

Definition:
WAP stands for **Wireless Application Protocol**. It is an **open international standard** developed to enable **mobile devices** such as mobile phones and PDAs to access **internet content** and services like email, news, and banking over **wireless networks**.

Objectives of WAP:

- Provide a **standard platform** for wireless communication.
- Support **wireless devices** with **limited resources** (memory, screen, power).
- Enable access to internet services using **mobile browsers**.

WAP Architecture:

```
java
CopyEdit
Mobile Device → WAP Gateway → Web Server (HTTP)
```

- The **WAP Gateway** acts as a bridge between wireless network and the Internet.
- Converts WML pages to binary for efficient transmission.

Key Components:

Layer	Description
WAE	Wireless Application Environment – equivalent to a mobile browser
WSP	Wireless Session Protocol – manages sessions
WTP	Wireless Transaction Protocol – ensures transaction reliability
WDP	Wireless Datagram Protocol – delivers data packets
Bearer Services	Underlying networks like GSM, CDMA, GPRS, etc.

Advantages:

- Low bandwidth usage
- Compatible with multiple wireless networks
- Device-independent protocol stack

Limitations:

- Very limited UI and design capabilities
- Largely outdated and replaced by **HTML5** and **mobile web apps**

Use Cases:

- Early mobile banking and messaging services
- Stock updates, news portals
- Location-based services before smartphones

✅ 2. WML (Wireless Markup Language)

[2 Marks – Structure, Syntax, Example]

Definition:
WML stands for **Wireless Markup Language**. It is a **markup language** developed specifically for **WAP-enabled mobile devices**. It is based on **XML**, and functions similarly to **HTML** but is optimized for the constraints of mobile phones.

Structure:

- A WML document is made up of **decks**, which are collections of **cards**.
- A **card** is like a web page screen; a **deck** is like a complete page or document.

Syntax:

- Strictly follows **XML syntax**.
- Must start with XML declaration and WML DTD.

Example:

```
xml
CopyEdit
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
 "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="home" title="Welcome">
    <p>Hello! This is a WML page.</p>
  </card>
</wml>
```

Features:

- Designed for small screens and low memory usage.
- Supports navigation, text input, and forms.
- Lightweight and easy to transmit over wireless networks.

Limitations:

- Cannot handle rich content like modern HTML.
- Very limited styling and layout options.
- Deprecated in favor of modern responsive web design.

✅ 3. .NET Framework

[4 Marks – Architecture, Components, Features, Uses]

Definition:
The **.NET Framework** is a **software development platform** developed by **Microsoft**. It provides a **runtime environment**, **libraries**, and **tools** for building and running Windows-based applications, web applications, and web services.

Main Components of .NET Framework:

a) Common Language Runtime (CLR):

- The **execution engine** of .NET applications.
- Converts MSIL (Microsoft Intermediate Language) to native code.
- Provides services like:
  - Memory management (Garbage Collection)
  - Exception handling
  - Type safety
  - Security

b) Framework Class Library (FCL):

- A rich set of pre-written **reusable classes** and **APIs**.
- Supports operations like:
  - File I/O
  - Database access (ADO.NET)
  - Web services
  - Collections, Strings, Threads

c) Languages Supported:

- C#, VB.NET, F#, and many others.
- All languages are compiled into **Common Intermediate Language (CIL)**.

d) ASP.NET:

- A subset used to build **dynamic web applications**.
- Offers Web Forms, MVC, Web API frameworks.

.NET Framework Architecture:

```
css
CopyEdit
[ Application Code ]
        ↓
[ Framework Class Library ]
        ↓
[ Common Language Runtime ]
        ↓
[ Operating System (Windows) ]
```

Key Features:

- **Cross-language integration** (C#, VB.NET, F#)
- **Object-oriented programming model**
- **Secure and robust environment**
- **Interoperability** with COM and native APIs
- **Managed code execution**

Advantages:

- Fast development and deployment with Visual Studio
- High-level security model
- Automatic memory management
- Enterprise-level scalability and performance

Use Cases:

- Enterprise resource planning (ERP) software
- E-commerce portals
- Web services and APIs
- Windows desktop applications

✅ Conclusion:

- **WAP** is a protocol that enabled mobile devices to connect to the internet during early mobile communication days.
- **WML** is the markup language used in WAP, specifically designed for small-screen devices.
- The **.NET Framework** is a modern, secure, and powerful platform for developing web and desktop applications on the Windows environment using managed code.

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

=>

✅ 1. Definition of WAP (Wireless Application Protocol)

[2 Marks]

WAP stands for **Wireless Application Protocol**. It is an **open global standard** developed to enable **wireless/mobile devices** like cell phones, PDAs, and early smartphones to access and interact with internet-based information and services.

It allows wireless devices to browse **WAP-enabled websites** that are written using **WML (Wireless Markup Language)** and optimized for low-bandwidth, high-latency environments.

✅ 2. Purpose and Need for WAP

- Traditional web content (HTML/CSS) was not suitable for mobile devices with small screens, low memory, and limited processing power.
- WAP provides a **lightweight, optimized protocol stack** similar to the internet, but designed for mobile environments.
- Helps in building applications like:
  - Mobile banking
  - News updates
  - Email
  - Stock tracking
  - Location-based services

✅ 3. Architecture of WAP

[6 Marks]

The WAP architecture is a layered protocol stack **similar to the OSI model** but optimized for mobile environments. It bridges mobile devices and internet services using a **WAP Gateway**.

Basic Flow:

```
java
CopyEdit
Mobile Device → WAP Gateway → Web Server (HTTP)
```

✅ 4. Components / Layers of WAP Architecture

Layer	Component Name	Description
5	**Wireless Application Environment (WAE)**	Equivalent to a mobile web browser; provides run-time environment for WML scripts and applications.
4	**Wireless Session Protocol (WSP)**	Manages sessions between client and server; similar to HTTP but optimized for wireless.
3	**Wireless Transaction Protocol (WTP)**	Handles request-response transactions; provides reliability and efficiency.
2	**Wireless Transport Layer Security (WTLS)**	Provides security (encryption, authentication, integrity) for wireless communication.
1	**Wireless Datagram Protocol (WDP)**	Adapts messages for different bearer networks like GSM, SMS, GPRS, CDMA.
-	**Bearer Services**	Underlying physical/mobile networks that carry data. (e.g., SMS, GSM, GPRS)

✅ Detailed Explanation of Each Layer:

1. WAE (Wireless Application Environment):

- Interface for developing and running wireless applications.
- Interprets WML and WMLScript.
- Similar to a simplified browser.

2. WSP (Wireless Session Protocol):

- Provides session management.
- Works like HTTP, but lighter.
- Supports persistent connections for efficiency.

3. WTP (Wireless Transaction Protocol):

- Ensures reliable data transmission.
- Offers three types of transactions:
  - Unreliable one-way
  - Reliable one-way
  - Reliable two-way

4. WTLS (Wireless TLS):

- Security layer between WTP and WDP.
- Provides encryption, authentication, and data integrity.
- Protects against data tampering and snooping.

5. WDP (Wireless Datagram Protocol):

- Transport layer protocol similar to UDP.
- Provides uniform interface to upper layers regardless of network type.

6. Bearer Networks:

- Physical/mobile network that delivers data.
- Examples: GSM, CDMA, SMS, GPRS, 3G

✅ 5. Diagram: (Optional for 8 marks – Good Impression)

```
pgsql
CopyEdit
+-------------------------------+
|     Wireless Applications     |
|           (WAE)               |
+-------------------------------+
|    Session Layer (WSP)        |
+-------------------------------+
|   Transaction Layer (WTP)     |
+-------------------------------+
|     Security Layer (WTLS)     |
+-------------------------------+
|   Transport Layer (WDP)       |
+-------------------------------+
|     Bearer Services (GSM, GPRS)|
+-------------------------------+
```

✅ 6. Advantages of WAP Architecture:

- Optimized for **low bandwidth and high latency**.
- Standardized communication between mobile and web.
- Secure, scalable, and lightweight.

✅ 7. Limitations of WAP:

- Outdated due to rise of HTML5 and modern responsive design.
- Limited UI capabilities.
- Replaced by smartphones and full-featured browsers.

✅ Conclusion:

WAP enabled early mobile devices to access internet content using a layered protocol architecture tailored for mobile limitations. Its modular design (WAE, WSP, WTP, WTLS, WDP) ensured efficient, secure, and reliable wireless communication.

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

=>

✅ 1. Definition of Multidimensional Array in PHP

[1 Mark]

A **multidimensional array** in PHP is an array **containing one or more arrays** as its elements. It allows the storage and organization of data in **table-like (matrix) format**, with **rows and columns**.

In simpler terms, it's an **array of arrays**, often used when data is structured in multiple dimensions — like a list of student records, each with multiple fields.

✅ 2. Types of Multidimensional Arrays

[1 Mark]

- **Two-dimensional array:**
  An array where each element is also an array — behaves like a table.
- **Three or more dimensions:**
  Arrays containing arrays that contain more arrays — used for complex data structures.

✅ 3. Example: Two-Dimensional Array in PHP

[2 Marks – Code + Output]

```
php
CopyEdit

<?php
// Multidimensional array of students
$students = array(
    array("John", 21, "B.Sc"),
    array("Alice", 22, "BCA"),
    array("Bob", 20, "BBA")
);

// Display student data using nested loop
for ($i = 0; $i < count($students); $i++) {
    echo "Student " . ($i+1) . ": ";
    for ($j = 0; $j < count($students[$i]); $j++) {
        echo $students[$i][$j] . " ";
    }
    echo "<br>";
}
?>
```

✅ 4. Output:

```
yaml
CopyEdit

Student 1: John 21 B.Sc
Student 2: Alice 22 BCA
Student 3: Bob 20 BBA
```

✅ 5. Explanation:

- The outer array `$students` holds three inner arrays (each student record).
- Each inner array contains **Name, Age, and Course**.
- Nested loops are used to access and display each value.

Q11. Explain overview of node JS. [5]

=>

✓ 1. Introduction to Node.js

[1 Mark]

Node.js is an **open-source**, **cross-platform**, **JavaScript runtime environment** that allows developers to run JavaScript **outside the web browser**, typically on the **server-side**.
It is built on **Google Chrome's V8 JavaScript engine**, and is known for its **non-blocking**, **event-driven** architecture.

✓ 2. Key Features of Node.js

[2 Marks]

Feature	Description
Asynchronous & Event-driven	All APIs are non-blocking. Node handles multiple requests simultaneously using event loop.
Single-threaded	Uses a single thread internally for handling requests efficiently.
Fast Execution	Built on V8 engine, which compiles JavaScript directly to native machine code.
No Buffering	Streams data in chunks; ideal for real-time applications like video/audio streaming.
Cross-platform	Runs on Windows, macOS, and Linux.

✓ 3. Node.js Architecture

[1 Mark]

- Uses a **single-threaded event loop**.
- Delegates tasks like file I/O, network operations to **libuv** and **worker threads**.
- Handles thousands of concurrent connections efficiently, making it suitable for real-time web apps.

✅ 4. Common Use Cases of Node.js

[1 Mark]

- Building **real-time applications** (e.g., chat apps, multiplayer games).
- **RESTful APIs** and **web services**.
- **Streaming servers** (Netflix, Spotify-like apps).
- **IoT device communication**.
- **Backend for Single Page Applications (SPAs)**.

✅ 5. Example Code (Optional for better understanding)

```
javascript
CopyEdit

// Simple Node.js server
const http = require('http');

http.createServer((req, res) => {
  res.write("Hello from Node.js!");
  res.end();
}).listen(3000);
```

✅ Conclusion:

Node.js provides a powerful environment to build **scalable, high-performance server-side applications** using JavaScript. Its non-blocking nature makes it ideal for **data-intensive and real-time applications**.

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

=>

5.15 Session Tracking

SPPU : Dec.-19, Marks 8

- When you open some application, use it for some time and then close it. This entire scenario is named as **session.**
- **Session state** is a server-based state mechanism that lets web applications store and retrieve objects of any type for each **unique user session.** That is, each browser session has its own session state stored as a serialized file on the server, which is deserialized and loaded into memory as needed for each request. Refer Fig. 5.15.1.

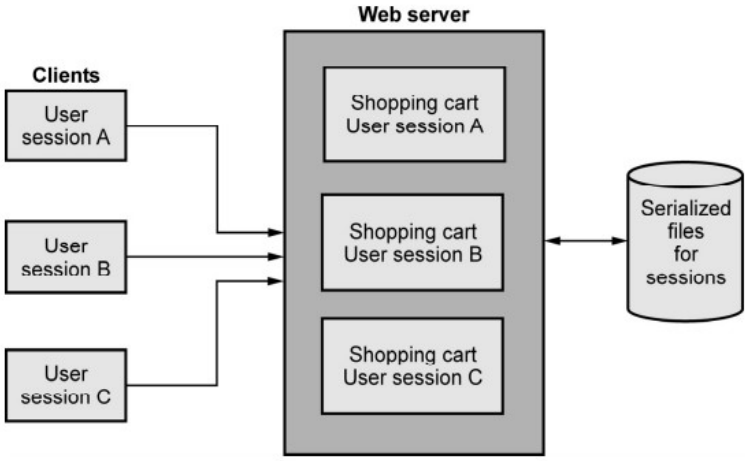


Fig. 5.15.1 Session state

✓ 1. What is Session Management in PHP?

[1 Mark]

Session management refers to the technique of **tracking user interactions** (such as login status, shopping cart contents, preferences, etc.) across multiple pages on a website.

Since HTTP is a **stateless protocol**, PHP uses **cookies** and **sessions** to maintain continuity between web pages during a user's visit.

✓ 2. What are Cookies in PHP?

[1 Mark]

- A **cookie** is a small piece of data stored **on the client-side (browser)**.
- Cookies are sent to the server with every HTTP request.
- They are used for storing **persistent data** like username, preferences, visit count, etc.

Syntax to Set a Cookie:

```
php
CopyEdit
setcookie("username", "John", time() + 3600); // expires in 1 hour
```

Accessing a Cookie:

```
php
CopyEdit
echo $_COOKIE["username"];
```

✅ 3. What are Sessions in PHP?

[1 Mark]

- A **session** stores data **on the server-side** and assigns a unique session ID to the user.
- The session ID is stored in the user's browser as a cookie named `PHPSESSID`.
- Ideal for **sensitive or temporary data** like user login information, cart items, etc.

Starting a Session:

```
php
CopyEdit
session_start(); // Must be called at the beginning of the script
$_SESSION["user"] = "John";
```

Accessing Session Data:

```
php
CopyEdit
echo $_SESSION["user"];
```

Destroying a Session:

```
php
CopyEdit
session_destroy();
```

✅ 4. Difference Between Cookies and Sessions

[2 Marks]

Feature	Cookies	Sessions
Storage	Stored in **client browser**	Stored on **server**
Security	Less secure (can be manipulated)	More secure
Data limit	~4 KB	No size limit (depends on server)
Lifetime	Can be set to persist	Expires when browser is closed or manually destroyed
Use Case	Store preferences, remember login	Handle login, cart, temp data

---

✅ 5. Example: Login using Session & Cookie

[2 Marks]

```
php
CopyEdit
// login.php
session_start();
$_SESSION["user"] = "admin";
setcookie("theme", "dark", time()+3600); // 1 hour
echo "Session and Cookie set.";
php
CopyEdit
// dashboard.php
session_start();
echo "Welcome, " . $_SESSION["user"];
echo "Theme: " . $_COOKIE["theme"];
```

✅ 6. Conclusion:

Both **cookies** and **sessions** play a key role in maintaining session data in PHP.

- **Cookies** store data **on the client** and are suitable for non-sensitive info.
- **Sessions** store data **on the server** and are more secure for tracking **logged-in users and temporary states**.

Together, they provide flexible and secure user session management in web applications.

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

=>

✅ 1. Definition of WML (Wireless Markup Language)

[1 Mark]

WML stands for **Wireless Markup Language**.
It is a **markup language** based on **XML** designed specifically for **WAP-enabled mobile devices** such as early mobile phones and PDAs.

WML was used to create pages (called **decks**) that could be rendered on small screens with **limited bandwidth, memory, and display capabilities**.

✅ 2. Structure of a WML Document

[0.5 Mark]

A typical WML document is structured as follows:

```
xml
CopyEdit
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
 "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="card1" title="Sample">
    <p>Hello, this is a WML example!</p>
  </card>
</wml>
```

✅ 3. WML Elements (Tags)

[3 Marks]

Here are the **important WML elements**:

Element	Description
`<wml>`	Root element of a WML document. All content must be within this tag.
`<card>`	Represents a single page or screen in a WML deck. Users can navigate between cards.
`<p>`	Paragraph element used to display text content.
`<anchor>`	Used to create hyperlinks within a card.
`<go>`	Defines an action (e.g., submitting a form or navigating to another card).

Element	Description
`<input>`	Accepts input from the user (like text, password, etc.).
`<select>`	Dropdown/select box for multiple choices.
`<option>`	Defines the individual items in a `<select>` list.
`<do>`	Specifies an action (like submit, navigate) associated with a card.

✅ 4. Example of WML with Elements

```
xml
CopyEdit
<wml>
  <card id="card1" title="Login">
    <p>Enter Username:</p>
    <input name="username" type="text"/>
    <do type="accept" label="Submit">
      <go href="welcome.wml"/>
    </do>
  </card>
</wml>
```

✅ 5. Conclusion

WML is an XML-based language used in the WAP framework to design web content for early mobile devices.
Its elements (like `<card>`, `<p>`, `<do>`, and `<input>`) allow developers to create structured, interactive content suitable for small screens and low bandwidth.

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

=>

✅ 1. Introduction to ASP.NET

[1 Mark]

ASP.NET is a **web development framework** developed by **Microsoft** for building **dynamic web applications**, websites, and web services.
It is a part of the **.NET Framework** and allows developers to build web apps using **C# or VB.NET**.

ASP.NET supports **server-side programming**, meaning the code is executed on the server before sending the output to the browser.

✅ 2. Key Features of ASP.NET

[2 Marks]

Feature	Description
Server-side technology	Code runs on the web server, improving security and performance.
Rich toolbox	Provides controls like textboxes, buttons, and data grids.
Compiled Code	ASP.NET code is compiled into DLLs for faster execution.
Language support	Supports multiple .NET languages, mainly C# and VB.NET.
State management	Built-in support for session, cookies, and view state.
Security	Offers built-in features for **authentication and authorization**.
MVC support	Supports **MVC architecture** for clean separation of concerns.

✅ 3. ASP.NET Application Lifecycle

[1 Mark]

The typical flow includes:

1. **Request** sent by the browser.
2. **IIS (Internet Information Services)** handles the request.
3. **ASP.NET runtime** processes the request.
4. Executes **page life cycle events** like `Page_Load()`, `Page_Init()`.
5. Response is **rendered as HTML** and sent back to the client.

✅ 4. Types of ASP.NET Applications

[0.5 Mark]

- **ASP.NET Web Forms** – Event-driven, drag-and-drop GUI-based development.
- **ASP.NET MVC** – Follows Model-View-Controller design pattern.
- **ASP.NET Core** – A cross-platform, high-performance framework for modern web apps.

✅ 5. Conclusion

[0.5 Mark]

ASP.NET is a powerful, scalable, and secure framework for building modern web applications.
It combines the robustness of the .NET platform with the flexibility of web technologies, making it ideal for enterprise-level websites and web services.

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

i) Arrays
ii) Function
iii) Control statements in PhP

=>

✅ i) Arrays in PHP

[3 Marks]

An **array** in PHP is a data structure used to **store multiple values** in a single variable. PHP supports **indexed**, **associative**, and **multidimensional arrays**.

1. Types of Arrays:

- **Indexed Array** – Stores values with numeric indexes.

```
php
CopyEdit

$colors = array("Red", "Green", "Blue");
echo $colors[1]; // Output: Green
```

- **Associative Array** – Uses named keys.

```
php
CopyEdit

$student = array("name"=>"Alice", "age"=>21);
echo $student["name"]; // Output: Alice
```

- **Multidimensional Array** – Array of arrays.

```
php
CopyEdit
```

```
$marks = array(
  array("Math", 90),
  array("Science", 85)
);
echo $marks[0][0]; // Output: Math
```

2. Useful Array Functions:

- `count()`, `array_push()`, `array_merge()`, `sort()`, `print_r()`, etc.

✅ ii) Functions in PHP

[3 Marks]

A **function** in PHP is a reusable block of code that performs a specific task.

1. Syntax:

```
php
CopyEdit

function greet($name) {
  return "Hello, " . $name;
}
echo greet("John"); // Output: Hello, John
```

2. Types of Functions:

- **Built-in functions**: Already provided by PHP (e.g., `strlen()`, `date()`, `array_merge()`).
- **User-defined functions**: Created by the programmer using `function` keyword.

3. Features:

- Can accept parameters and return values.
- Promotes code **modularity** and **reusability**.
- Supports **default** and **optional** arguments.

```
php
CopyEdit

function add($a, $b=5) {
  return $a + $b;
}
echo add(10); // Output: 15
```

✅ iii) Control Statements in PHP

[3 Marks]

Control statements allow **decision-making** and **looping** in PHP programs.

1. Conditional Statements:

- `if`, `if-else`, `if-elseif-else`, `switch`

```
php
CopyEdit

$age = 18;
if ($age >= 18) {
  echo "Adult";
} else {
  echo "Minor";
}
```

2. Looping Statements:

- `for`, `while`, `do-while`, `foreach`

```
php
CopyEdit

for ($i = 1; $i <= 5; $i++) {
  echo $i . " ";
}
```

- `foreach` – Specially used for arrays.

```
php
CopyEdit

$fruits = array("apple", "banana");
foreach ($fruits as $fruit) {
  echo $fruit . " ";
}
```

3. Jump Statements:

- `break`, `continue`, `return`

```
php
CopyEdit

for ($i = 1; $i <= 5; $i++) {
  if ($i == 3) continue;
  echo $i . " ";
}
// Output: 1 2 4 5
```

✅ Conclusion:

PHP provides powerful constructs like **arrays**, **functions**, and **control statements** to build dynamic, efficient, and flexible web applications. Understanding these is essential for logical program design and flow control.

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

=>>

✅ 1. Introduction to Arrays in PHP

[1 Mark]

An **array** in PHP is a **collection of elements** stored under a single variable name. Each element is stored as a key-value pair, and arrays are used to store **multiple values** in a structured way.

PHP arrays are flexible and can hold any data type — integers, strings, objects, even other arrays.

✅ 2. Types of Arrays in PHP

[2 Marks]

PHP supports **three types of arrays**:

a) Indexed Array

- Elements are indexed with **numeric keys (0, 1, 2, ...)**.

```
php
CopyEdit

$fruits = array("Apple", "Banana", "Mango");
echo $fruits[1];  // Output: Banana
```

b) Associative Array

- Elements are stored using **named keys**.

```
php
CopyEdit

$student = array("name" => "Ravi", "age" => 21, "course" => "BCA");
echo $student["course"];  // Output: BCA
```

c) Multidimensional Array

- An array that contains **one or more arrays** inside it.

```
php
CopyEdit

$marks = array(
    "Ravi" => array("Math" => 90, "Science" => 85),
    "Sita" => array("Math" => 88, "Science" => 92)
);
echo $marks["Ravi"]["Science"];  // Output: 85
```

✅ 3. Creating and Using Arrays

[2 Marks]

a) Declaring an Array:

```
php
CopyEdit

$colors = array("Red", "Green", "Blue");
```

b) Adding Elements:

```
php
CopyEdit

$colors[] = "Yellow";  // Adds Yellow to the array
```

c) Accessing Elements:

```
php
CopyEdit

echo $colors[0];  // Output: Red
```

d) Modifying Values:

```
php
CopyEdit

$colors[1] = "Purple";  // Replaces Green with Purple
```

✅ 4. Useful Array Functions

[2 Marks]

PHP provides many built-in functions to manipulate arrays:

Function	Purpose
`count($arr)`	Returns number of elements
`array_push($arr, $val)`	Adds value to end of array
`array_merge($a, $b)`	Merges two arrays
`sort($arr)`	Sorts array in ascending order
`print_r($arr)`	Prints readable structure of array

Example:

```
php
CopyEdit

$numbers = array(10, 5, 7);
sort($numbers);
print_r($numbers);
// Output: Array ( [0] => 5 [1] => 7 [2] => 10 )
```

✅ 5. Looping Through Arrays

[1 Mark]

Use `foreach` loop to iterate through array elements:

```
php
CopyEdit

$names = array("Amit", "Raj", "Neha");

foreach ($names as $name) {
    echo $name . "<br>";
}
```

Output:

```
nginx
CopyEdit

Amit
Raj
Neha
```

✅ 6. Conclusion

Arrays in PHP are a powerful feature used to handle **collections of related data**. They support various types (indexed, associative, multidimensional) and are backed by a rich set of built-in functions for sorting, merging, counting, and more — making data manipulation simple and efficient.

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

=>

✅ 1. Introduction to OOP in PHP with MySQL

[1 Mark]

PHP provides two ways to connect to a MySQL database:

1. **Procedural style** using `mysqli_connect()`
2. **Object-oriented style** using the **`mysqli` class**

The object-oriented (OOP) way is **more structured, reusable, and secure**, and it makes use of **classes and objects** to handle database operations.

✅ 2. Steps to Connect to MySQL Using OOP Style

[2 Marks]

To use the object-oriented style:

1. Create an object of the `mysqli` class.
2. Use methods like `connect_error`, `query()`, `fetch_assoc()`.

Basic Syntax:

```
php
CopyEdit

$conn = new mysqli(servername, username, password, database);
```

✅ 3. Example: Connect and Display Data

[3 Marks]

```
php
CopyEdit
```

```php
<?php
// Step 1: Database connection parameters
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "college";

// Step 2: Create connection using OOP
$conn = new mysqli($servername, $username, $password, $dbname);

// Step 3: Check for connection errors
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

// Step 4: Write SQL query and execute it
$sql = "SELECT id, name, department FROM students";
$result = $conn->query($sql);

// Step 5: Check if rows are returned
if ($result->num_rows > 0) {
    // Step 6: Output data using fetch_assoc()
    while($row = $result->fetch_assoc()) {
        echo "ID: " . $row["id"] . " - Name: " . $row["name"] .
             " - Dept: " . $row["department"] . "<br>";
    }
} else {
    echo "No records found.";
}

// Step 7: Close connection
$conn->close();
?>
```

✅ 4. Explanation of Key OOP Methods

[2 Marks]

Method/Property	Description
`new mysqli()`	Creates a new connection object.
`$conn->connect_error`	Checks if connection failed.
`$conn->query($sql)`	Executes the SQL query.
`$result->fetch_assoc()`	Fetches each row as an associative array.
`$conn->close()`	Closes the database connection.

✅ 5. Conclusion

Object-oriented database connectivity in PHP using the **`mysqli` class** is a **clean and modular** approach.
It allows better error handling, easier debugging, and follows modern programming standards.
It is preferred for larger or more secure PHP applications.

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

=>

✅ 1. Introduction to .NET Framework

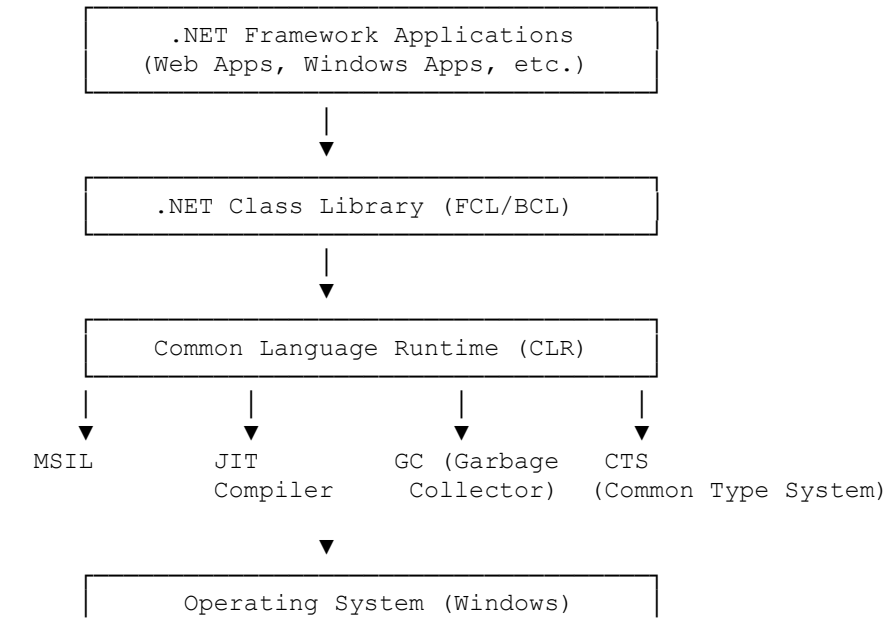
[1 Mark]

The **.NET Framework** is a **software development platform** developed by **Microsoft**. It provides a **runtime environment** and a set of **libraries and tools** to develop applications that can run on Windows.
It supports multiple languages like **C#, VB.NET, F#**, etc., and allows language interoperability.

✅ 2. Neat Diagram of .NET Framework Architecture

[2 Marks]

```
scss
CopyEdit
```



└_____________________________________┘

✓ 3. Key Components of .NET Framework

[4 Marks]

i) CLR (Common Language Runtime)

- It is the **execution engine** of the .NET Framework.
- Converts **MSIL (Microsoft Intermediate Language)** into **native machine code** using **JIT (Just-In-Time) compiler**.
- Provides services like:
  - **Memory management**
  - **Garbage collection**
  - **Exception handling**
  - **Security** and **type safety**

ii) FCL/BCL (Framework Class Library / Base Class Library)

- A large set of **predefined reusable classes and interfaces**.
- Provides built-in support for **file handling**, **input/output**, **collections**, **database access (ADO.NET)**, **networking**, and more.

iii) CTS (Common Type System)

- Defines a set of **data types** and rules so that objects written in different .NET languages can **interact with each other**.

iv) CLS (Common Language Specification)

- A set of **rules and standards** for writing code that can be used across all .NET languages.
- Ensures **language interoperability**.

v) JIT Compiler

- Part of CLR.
- Converts **MSIL code** to **native machine code** just before execution.

vi) Garbage Collector (GC)

- Automatically **manages memory** by removing objects that are no longer used.

✓ 4. CLI (Common Language Infrastructure)

[1.5 Marks]

- CLI is an **open standard** developed by Microsoft and standardized by **ECMA** and **ISO**.
- It defines the **runtime environment**, including:
  - **MSIL code**
  - **Type system**
  - **Metadata format**
  - **Execution engine**
- The **.NET Framework** is Microsoft's implementation of the CLI.

✅ 5. Conclusion

[0.5 Mark]

The **.NET Framework** simplifies application development by offering a **powerful runtime (CLR)** and a **rich class library**.
With **CLR** and **CLI**, it ensures **language independence**, **performance**, and **secure execution** of applications.

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

=>

✅ 1. Introduction

[1 Mark]

PHP allows connecting to a MySQL database using **MySQLi** (MySQL improved) or **PDO (PHP Data Objects)**.
This connection enables PHP to interact with the database for **inserting**, **retrieving**, **updating**, or **deleting** data.

There are two styles for connecting:

- **Procedural style** (using functions)
- **Object-Oriented style** (using classes and objects)

This answer explains using the **procedural style with MySQLi**.

✅ 2. Steps to Connect to MySQL Database Using PHP

[6 Marks]

Here are the **step-by-step procedures** with explanation and code snippets:

Step 1: Define Database Connection Parameters

```
php
CopyEdit
$host = "localhost";
$username = "root";
$password = "";
$database = "college";
```

- `localhost` = server location
- `root` = default MySQL username
- `""` = blank password (default)
- `"college"` = name of the database

Step 2: Create Connection

```
php
CopyEdit
$conn = mysqli_connect($host, $username, $password, $database);
```

- `mysqli_connect()` is a built-in PHP function to connect to MySQL.
- It returns a **connection object** if successful or **false** on failure.

Step 3: Check the Connection

```
php
CopyEdit
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
} else {
    echo "Connected successfully";
}
```

- `mysqli_connect_error()` gives the reason for connection failure.

Step 4: Execute SQL Query

```
php
CopyEdit
$sql = "SELECT * FROM students";
```

```
$result = mysqli_query($conn, $sql);
```

- `mysqli_query()` executes the SQL query and stores the result.

Step 5: Fetch and Display Results

```
php
CopyEdit
if (mysqli_num_rows($result) > 0) {
    while($row = mysqli_fetch_assoc($result)) {
        echo "ID: ".$row["id"]." - Name: ".$row["name"]."<br>";
    }
} else {
    echo "No records found.";
}
```

- `mysqli_fetch_assoc()` returns each row as an **associative array**.

Step 6: Close the Connection

```
php
CopyEdit
mysqli_close($conn);
```

- Always close the connection after operations are complete.

✅ 3. Complete Example Code

[1 Mark]

```
php
CopyEdit
<?php
$host = "localhost";
$username = "root";
$password = "";
$database = "college";

// Step 1 and 2: Connect
$conn = mysqli_connect($host, $username, $password, $database);

// Step 3: Check connection
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Step 4: Execute query
$sql = "SELECT * FROM students";
```

```php
$result = mysqli_query($conn, $sql);

// Step 5: Fetch and display
if (mysqli_num_rows($result) > 0) {
    while($row = mysqli_fetch_assoc($result)) {
        echo "ID: " . $row["id"] . " - Name: " . $row["name"] . "<br>";
    }
} else {
    echo "No data found.";
}

// Step 6: Close connection
mysqli_close($conn);
?>
```

✅ 4. Conclusion

[1 Mark]

Connecting PHP to MySQL involves steps like defining connection parameters, establishing connection using `mysqli_connect()`, executing queries, fetching results, and closing the connection. This is essential for **database-driven applications** like login systems, content management, and data dashboards.

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

=>

✅ i) Overview of ASP.NET

[4.5 Marks]

1. Introduction

ASP.NET is a **web development framework** developed by **Microsoft** to build **dynamic web applications**, **web services**, and **web APIs**.

2. Key Features

- Built on the **.NET Framework** (or now on **.NET Core** / **.NET 5/6+**).
- Supports **server-side programming**.
- Enables code written in **C#, VB.NET**, etc.
- Offers **Web Forms**, **MVC**, **Razor Pages**, and **Blazor** for web UI design.

3. Advantages

- **Separation of concerns** using MVC model.
- **State management** via ViewState, Session, Cookies.

- **Built-in security**: Authentication & Authorization.
- **Rich Toolbox** in Visual Studio (drag and drop controls).
- **Event-driven programming model**.

4. Example

```
asp
CopyEdit
<%@ Page Language="C#" %>
<html>
<body>
  <form runat="server">
    <asp:Button ID="btnClick" runat="server" Text="Click Me"
OnClick="btnClick_Click" />
  </form>
</body>
</html>
```

✅ ii) Overview of C# (C-Sharp)

[4.5 Marks]

1. Introduction

C# is a **modern, object-oriented programming language** developed by **Microsoft** under the .NET platform.
It is simple, type-safe, and scalable for building applications ranging from desktop to web and cloud.

2. Features of C#

- **Object-Oriented**: Supports classes, inheritance, polymorphism.
- **Type-safe**: Detects type errors at compile time.
- **Interoperable**: Works with other .NET languages.
- **Automatic Garbage Collection**
- Supports **LINQ**, **Asynchronous Programming**, **Events**, and **Delegates**.

3. Applications

- Desktop applications (using Windows Forms or WPF)
- Web applications (with ASP.NET)
- Game development (with Unity engine)
- Mobile apps (via Xamarin/.NET MAUI)

4. Simple Example

```
csharp
CopyEdit
using System;

class HelloWorld {
```

```
    static void Main() {
        Console.WriteLine("Hello, C# World!");
    }
}
```

✅ Conclusion

- **ASP.NET** is a powerful framework for developing web-based applications with support for reusable components and modern web practices.
- **C#** is the backbone language of the .NET ecosystem, offering strong features for general-purpose and enterprise-level programming.

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

=>

- The WAP model follows the OSI model and is represented by Fig. 5.17.1.

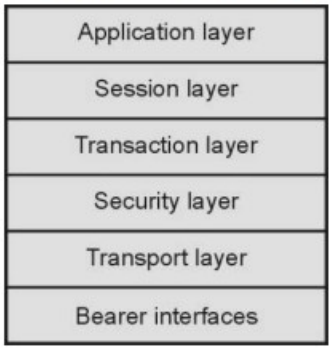


Fig. 5.17.1 WAP model

✅ 1. Introduction to WAP

[1 Mark]

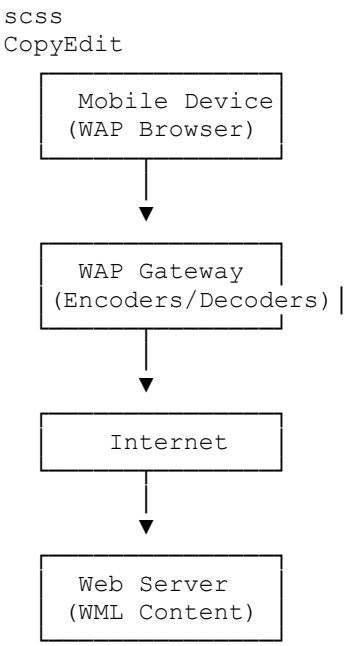
- **WAP (Wireless Application Protocol)** is a **technical standard** developed to enable **mobile devices** (like early phones and PDAs) to access the **Internet**.
- It allows browsing **text-based web pages** on devices with small screens, **limited bandwidth**, and **low processing power**.

✅ 2. WAP Architecture

[4 Marks]

WAP architecture is designed to be **lightweight, efficient, and optimized** for wireless communication.

Diagram of WAP Architecture:



Main Components of WAP Architecture:

1. **Mobile Device with WAP Browser:**
   - Contains a **micro-browser** that understands **WML**.
   - Sends requests to the WAP Gateway.
2. **WAP Gateway (or WAP Proxy):**
   - Acts as a **bridge between mobile network and the Internet**.
   - Performs:
     - **Encoding/Decoding** (WML to HTML or vice versa)
     - **Content translation**
     - **Compression** for data optimization
3. **Web Server:**
   - Hosts the WAP content in **WML format**.
   - Returns **WML pages** in response to requests.
4. **Internet:**
   - Standard TCP/IP-based network used to communicate with web servers.

✅ 3. WML (Wireless Markup Language)

5.18 WML

- The topmost layer in the WAP architecture is made up of WAE (Wireless Application Environment), which consists of WML and WML scripting language.
- WML stands for Wireless Markup Language.
- The role of WML is the same as that of HTML in web applications. WAP sites are written in WML, while web sites are written in HTML.
- WML is basically the web language for making sites on mobile phones.
- WML is an application of XML, which is defined in a document-type definition.

How to write WML ?

- The WML is a scripting language just similar to HTML. The web script can be written in Notepad and executed on WAP simulator.
- The extension to the WML file is .wml.
- I have used WinWap for executing the WML scripts. The WinWAP is a web browser for WAP made by Winwap Technologies available for Microsoft Windows. Get it downloaded from internet using the site www.winwap.com, if you wish to use WinWap browser for executing the WML scripts.

[3 Marks]

What is WML?

- WML is the **markup language** used to write **WAP applications**.
- It is **similar to HTML** but **optimized for small screens** and low bandwidth.
- WML is based on **XML**.

Structure of WML

- WML content is organized into **decks** and **cards**:
  - A **deck** is a WML file.
  - A **card** is a single screen of content in a mobile browser.

Basic Syntax Example:

```
xml
```

```
CopyEdit

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
   "http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
  <card id="welcome" title="Welcome Page">
    <p>Welcome to WAP world!</p>
  </card>
</wml>
```

Key WML Elements:

Element	Description
`<wml>`	Root element
`<card>`	Represents a single user interaction screen
`<p>`	Used for paragraphs or text
`<do>`	Handles user actions (like submit, go)
`<input>`	To accept user input

✅ 4. Conclusion

[1 Mark]

WAP enables wireless devices to access the internet using a **lightweight protocol** and **WML** for content delivery.
Its architecture bridges mobile networks and web content through a **gateway**, making it essential for early mobile browsing before modern mobile web standards emerged.

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

=>

✅ Part A: Functions in PHP

[4.5 Marks]

1. What is a Function?

- A **function** in PHP is a block of code that performs a specific task.
- It allows **code reusability**, **modularity**, and **easier debugging**.

2. Types of Functions in PHP:

- **Built-in functions:** Already available in PHP (e.g., `strlen()`, `date()`)
- **User-defined functions:** Created by the programmer

3. Syntax of User-defined Function:

```
php
CopyEdit

function functionName($param1, $param2) {
    // Code block
    return $result;
}
```

4. Example of a Function:

```
php
CopyEdit

<?php
function addNumbers($a, $b) {
    $sum = $a + $b;
    return $sum;
}

$result = addNumbers(5, 10);
echo "Sum is: " . $result;
?>
```

Output:

```
Sum is: 15
```

5. Benefits:

- Avoids code repetition
- Makes code cleaner and organized
- Easier to debug and test modules independently

✅ Part B: Session Management in PHP

[4.5 Marks]

1. What is a Session?

- A **session** is a way to store information (in variables) to be used across multiple pages.

- Unlike cookies, the session information is **stored on the server**, making it **more secure**.

2. Starting a Session

Use `session_start()` before using session variables.

```
php
CopyEdit

<?php
session_start(); // Must be called at the beginning of the script
?>
```

---

3. Storing Data in Session

```
php
CopyEdit

<?php
session_start();
$_SESSION["username"] = "JohnDoe";
echo "Session set for user: " . $_SESSION["username"];
?>
```

---

4. Retrieving Session Data

```
php
CopyEdit

<?php
session_start();
echo "Welcome, " . $_SESSION["username"];
?>
```

---

5. Destroying a Session

```
php
CopyEdit

<?php
session_start();
session_destroy();
echo "Session ended.";
?>
```

---

6. Use Cases:

- Login/logout systems
- Shopping carts

- User preferences or settings

✅ Conclusion

- **Functions in PHP** are essential for modular and reusable code.
- **Session management** allows developers to preserve user data across web pages securely, enabling personalized and stateful web applications.

UNIT 6

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

=>

1. Introduction to Ruby

[2 Marks]

Ruby is a **high-level**, **interpreted**, **general-purpose programming language** designed by Yukihiro Matsumoto (also known as Matz) in **1995**.
It is **object-oriented**, meaning everything in Ruby is treated as an object, including primitive data types like numbers and strings.
Ruby was created to make programming enjoyable, emphasizing **simplicity, productivity, and elegant syntax**.

---

2. Advantages of Ruby

[4 Marks]

- **Pure Object-Oriented:**
  Everything in Ruby is an object, even basic data types like integers and booleans. This consistency makes Ruby easy to learn and use for object-oriented programming.
- **Simple and Readable Syntax:**
  Ruby code reads almost like English, making it easy for beginners and increasing code maintainability.
- **Dynamic Typing and Duck Typing:**
  Variables do not require explicit types; types are checked at runtime. Duck typing focuses on whether an object supports a method rather than its class type, which promotes flexibility.

- **Garbage Collection:**
  Ruby automatically manages memory allocation and deallocation, so developers don't need to worry about memory leaks.
- **Rich Standard Library & Gems:**
  Ruby comes with a vast standard library and a large repository of external libraries called **gems**, which simplify adding functionalities like web development, testing, and database access.
- **Cross-Platform:**
  Ruby runs on multiple platforms including Windows, Linux, and macOS without changes in code.
- **Flexible & Expressive:**
  Ruby supports multiple programming paradigms such as procedural, functional, and imperative programming, offering programmers flexibility.
- **Active Community & Frameworks:**
  Ruby has an active developer community and powerful frameworks like **Ruby on Rails** that make web development faster and easier.

---

3. Control Statements in Ruby

[4 Marks]

Control statements guide the program's flow based on conditions and loops. Ruby provides a rich set of control flow constructs:

---

a) if...else Statement:

Used to execute code blocks conditionally.

```
ruby
CopyEdit
age = 20
if age >= 18
  puts "You are eligible to vote."
else
  puts "You are not eligible."
end
```

- The `if` block executes if the condition is true.
- The `else` block executes if the condition is false.

---

b) elsif Statement:

Used for multiple conditions.

```
ruby
CopyEdit
marks = 85

if marks >= 90
  puts "Grade A"
elsif marks >= 75
  puts "Grade B"
else
  puts "Grade C"
end
```

---

c) unless Statement:

Opposite of `if` — executes when the condition is false.

```
ruby
CopyEdit
logged_in = false

unless logged_in
  puts "Please login first."
end
```

---

d) case Statement:

Acts like a switch-case in other languages, useful for multiple discrete conditions.

```
ruby
CopyEdit
day = "Friday"

case day
when "Monday"
  puts "Start of the week"
when "Friday"
  puts "End of the week"
else
  puts "Midweek"
end
```

---

e) Loops:

To repeat blocks of code.

- **while loop:**

```
ruby
CopyEdit
i = 1
while i <= 5
  puts i
```

```
  i += 1
end
```

- **for loop:**

```
ruby
CopyEdit

for i in 1..5
  puts i
end
```

- **each iterator (more Ruby idiomatic):**

```
ruby
CopyEdit

[1, 2, 3, 4, 5].each do |num|
  puts num
end
```

---

f) Loop Control Keywords:

- `break` — exits the loop immediately.
- `next` — skips to the next iteration.
- `redo` — repeats the current iteration without reevaluating the condition.

Example:

```
ruby
CopyEdit

i = 0
while i < 5
  i += 1
  next if i == 3
  puts i
end
```

(Output skips printing 3)

---

4. Summary

- Ruby's **object-oriented design** and **clean syntax** make it an ideal choice for beginners and professionals.
- Its control statements allow flexible, readable, and efficient flow control.
- Ruby's support for different control structures and loop types helps in writing expressive and concise code.
- These features combined with powerful frameworks have made Ruby a popular language in the software development industry.

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

=>

1. What is EJB?

(3 Marks)

- **EJB (Enterprise JavaBeans)** is a **server-side component architecture** for building scalable, distributed, and transactional business applications in Java.
- It is part of the **Java EE (Enterprise Edition)** platform, designed to simplify the development of large-scale enterprise applications by handling system-level concerns like transactions, security, and remote access.
- EJB components run inside an **EJB container** which provides services such as **transaction management**, **security**, **concurrency**, and **lifecycle management**.
- Types of EJBs:
  - **Session Beans** (Stateless, Stateful, Singleton)
  - **Message-Driven Beans (MDB)**

---

2. Key Features of EJB:

- Simplifies complex enterprise application development
- Provides built-in support for transactions and security
- Supports remote method invocation
- Supports concurrency and pooling
- Integrates easily with databases and other Java EE components

---

3. Five Basic Examples of Using EJB

(4 Marks)

Here are five typical use cases/examples where EJB is commonly used:

---

a) Stateless Session Bean for Business Logic

A bean that performs a stateless business operation, e.g., calculating discounts.

```
java
CopyEdit
@Stateless
```

```
public class DiscountBean {
    public double calculateDiscount(double price) {
        return price * 0.10; // 10% discount
    }
}
```

---

b) Stateful Session Bean for Shopping Cart

Maintains user session state, such as items in a shopping cart.

```
java
CopyEdit
@Stateful
public class ShoppingCartBean {
    private List<String> items = new ArrayList<>();

    public void addItem(String item) {
        items.add(item);
    }

    public List<String> getItems() {
        return items;
    }
}
```

---

c) Singleton Bean for Application-wide Cache

Used to share state or cache data across the entire application.

```
java
CopyEdit
@Singleton
public class CacheBean {
    private Map<String, String> cache = new HashMap<>();

    public void put(String key, String value) {
        cache.put(key, value);
    }

    public String get(String key) {
        return cache.get(key);
    }
}
```

---

d) Message-Driven Bean (MDB) for Asynchronous Processing

Handles messages from JMS queues/topics, such as processing orders asynchronously.

```
java
CopyEdit
@MessageDriven(activationConfig = {
```

```
  @ActivationConfigProperty(propertyName = "destinationType", propertyValue
= "javax.jms.Queue")
})
public class OrderProcessorBean implements MessageListener {
    public void onMessage(Message message) {
        // process order message
    }
}
```

---

e) EJB Timer Service for Scheduling Tasks

Scheduling automatic periodic tasks like generating reports.

```
java
CopyEdit
@Stateless
public class ReportBean {

    @Schedule(hour="0", minute="0", persistent=false)
    public void generateDailyReport() {
        // generate report logic
    }
}
```

---

4. Conclusion

EJB simplifies enterprise application development by abstracting complex system-level services. Using EJBs, developers can focus on business logic while the container manages transactions, security, and scalability.

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

=>

✓ Part 1: Arrays in Ruby

[5 Marks]

1. What is an Array?

An **array in Ruby** is an **ordered collection of elements**. It can hold objects of **any data type**, including strings, integers, other arrays, or even custom objects.

```
ruby
CopyEdit


arr = [1, "two", 3.0, [4, 5], true]
```

Ruby arrays are dynamic and **can grow or shrink** as needed.

---

2. Array Creation:

```
ruby
CopyEdit

# Empty array
a = Array.new

# With elements
b = [1, 2, 3, 4]
```

---

3. Common Array Methods:

Method	Description
`push` or `<<`	Adds element to end
`pop`	Removes last element
`shift`	Removes first element
`unshift`	Adds element at the beginning
`length`	Returns number of elements
`each`	Iterates over elements
`map`, `select`	Transformation and filtering
`include?`	Checks if element is present

```
ruby
CopyEdit

numbers = [1, 2, 3, 4]
numbers.push(5)        # => [1, 2, 3, 4, 5]
numbers.include?(3)    # => true
numbers.each { |n| puts n }
```

---

4. Iterating Over Arrays:

```
ruby
CopyEdit

fruits = ["apple", "banana", "cherry"]
fruits.each do |fruit|
  puts "I like #{fruit}"
end
```

---

5. Multidimensional Arrays:

```
ruby
CopyEdit

matrix = [[1, 2], [3, 4], [5, 6]]
puts matrix[1][1]  # Output: 4
```

✅ Part 2: Rails with AJAX

[5 Marks]

1. What is AJAX?

AJAX stands for **Asynchronous JavaScript and XML**. It allows **web pages to update asynchronously** by exchanging data with the server **without refreshing the whole page**.

---

2. Why Use AJAX in Rails?

Rails provides **built-in support for AJAX** through unobtrusive JavaScript (UJS). It helps create a **faster, more interactive UI** without reloading pages.

---

3. How AJAX Works in Rails:

Workflow:

1. User triggers an event (click or form submit).
2. Rails sends an AJAX request (JavaScript format).
3. Controller processes it and returns JS/JSON response.
4. JavaScript response is executed to update part of the page.

---

4. Example: AJAX in Rails with Remote Form

View (ERB):

```
erb
CopyEdit

<%= form_with(url: books_path, remote: true) do |f| %>
```

```
  <%= f.text_field :title %>
  <%= f.submit "Add Book" %>
<% end %>
```

Controller:

```
ruby
CopyEdit

def create
  @book = Book.new(book_params)
  if @book.save
    respond_to do |format|
      format.js   # create.js.erb
    end
  end
end
```

create.js.erb (AJAX response):

```
js
CopyEdit
$("#book_list").append("<li><%= j @book.title %></li>");
```

---

5. Benefits of AJAX in Rails:

- Improves **user experience** by avoiding full page reloads.
- Easy to implement using **form_with**, `remote: true`, and `.js.erb` views.
- Makes applications **faster and more dynamic**.

✅ Conclusion:

Ruby arrays are powerful and flexible data structures used in daily programming. Rails supports AJAX natively, allowing developers to build responsive, user-friendly applications that interact with the server **seamlessly and asynchronously**.

Q4. Explain Document Request in Rails. [4]

=>

✅ What is a Document Request in Rails?

In Ruby on Rails, a **document request** refers to the **HTTP request** made by the browser (or client) to the Rails server to **fetch a document or resource** such as an HTML page, JSON data, XML response, or JavaScript file.

Rails handles different **MIME types** (content types) using a feature called **respond_to**. Based on the request format, Rails decides what kind of document to return.

✅ Types of Document Requests:

Rails can respond to multiple formats:

- `text/html` – for standard web pages
- `application/json` – for APIs or AJAX responses
- `application/xml` – for XML data (less common)
- `text/javascript` – for AJAX requests returning JavaScript

✅ Example: Handling Different Document Requests in Rails

```
ruby
CopyEdit

def show
  @book = Book.find(params[:id])

  respond_to do |format|
    format.html  # show.html.erb
    format.json { render json: @book }
    format.xml  { render xml: @book }
  end
end
```

In the above example:

- If the browser requests HTML, Rails renders `show.html.erb`.
- If the request is from an API or JavaScript with `.json`, it returns a JSON document.
- If the request format is XML, it renders XML.

✅ How Document Request Works:

1. **Browser/User sends a request** with an "Accept" header (e.g., Accept: application/json).
2. **Rails controller reads the request format.**
3. **`respond_to` block** in controller decides which template or response to send.
4. Rails sends the correct document back (HTML/JSON/XML).

✅ Conclusion:

Document requests in Rails allow the application to deliver content in different formats depending on the client's needs. This is essential for building **web pages, APIs, and AJAX-based applications** in a clean and scalable way.

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

=>

✅ 1. Advantages of Ruby (the Language):

- **Simple and Readable Syntax:**
  Ruby uses plain, human-readable code that looks like English, making it easy to write and understand.
- **Object-Oriented:**
  Everything in Ruby is an object, allowing better organization and code reuse through classes and methods.
- **Dynamic and Flexible:**
  Ruby allows dynamic typing and supports metaprogramming, making development faster and more adaptable.

✅ 2. Advantages of Ruby on Rails (the Framework):

- **Convention over Configuration:**
  Rails follows smart defaults and naming conventions, reducing the need to write boilerplate code.
- **Rapid Development:**
  Built-in tools like scaffolding and generators help developers create web applications very quickly.
- **Built-in Features:**
  Rails provides powerful features like routing, Active Record ORM, session management, and integrated testing tools.
- **Strong Community & Libraries:**
  A large open-source community and many gems (plugins) are available for adding functionality easily.

✅ Conclusion:

Ruby offers simplicity and flexibility as a language, while **Rails** enhances productivity and structure in web development, making them ideal for building modern web applications quickly and efficiently.

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

=>

✅ 1. Scalar Types in Ruby

[3 Marks]

Scalar types are basic data types that hold a single value. In Ruby, the main scalar types are:

a) Integer

- Whole numbers (positive or negative)

```
ruby
CopyEdit

a = 10
b = -5
```

b) Float

- Decimal numbers

```
ruby
CopyEdit

price = 99.99
```

c) String

- A sequence of characters

```
ruby
CopyEdit

name = "Ruby"
```

d) Boolean

- Represents `true` or `false`

```
ruby
CopyEdit

is_valid = true
```

e) Symbol

- Lightweight string-like identifiers, often used as keys or identifiers

```
ruby
CopyEdit

status = :active
```

f) Nil

- Represents “nothing” or “no value”

```
ruby
CopyEdit

user = nil
```

✅ 2. Operations in Ruby

[3 Marks]

Ruby supports several operations on scalar types:

a) Arithmetic Operations (on Integer & Float):

```
ruby
CopyEdit

a = 10
b = 5
puts a + b     # 15
puts a * b     # 50
puts a / b     # 2
puts a % b     # 0
```

b) String Operations:

```
ruby
CopyEdit

greeting = "Hello " + "World"  # "Hello World"
repeat = "Ruby" * 3           # "RubyRubyRuby"
```

c) Boolean Logic Operations:

```
ruby
CopyEdit

puts true && false   # false
puts true || false   # true
puts !true           # false
```

d) Comparison Operations:

```
ruby
CopyEdit

puts 5 == 5        # true
puts 10 > 7        # true
puts "a" < "b"     # true
```

✅ 3. Pattern Matching in Ruby

[3 Marks]

Pattern Matching is a powerful feature introduced in **Ruby 2.7** and improved in **Ruby 3.0**. It allows matching structured data in a clean and readable way using the `case` or `in` keywords.

a) Basic Pattern Matching:

```
ruby
CopyEdit

case [1, 2, 3]
in [a, b, c]
  puts a      # Output: 1
end
```

b) Pattern Matching with Hashes:

```
ruby
CopyEdit

user = {name: "Alice", age: 22}

case user
in {name: "Alice", age: age}
  puts age    # Output: 22
end
```

c) Using Pattern Matching with => Operator:

```
ruby
CopyEdit

data = [10, 20]

case data
in [x, y]
  puts "x = #{x}, y = #{y}"   # x = 10, y = 20
end
```

d) Pattern Matching with Guard Conditions:

```
ruby
CopyEdit
```

```
person = {name: "Bob", age: 30}

case person
in {age: age} if age > 25
  puts "Adult"
end
```

✅ Conclusion:

- Ruby's **scalar types** represent basic values like numbers, strings, and booleans.
- It supports **powerful operations** like arithmetic, logical, and string manipulation.
- With **pattern matching**, Ruby allows elegant destructuring and condition-based handling of arrays, hashes, and objects—making code cleaner and more readable.

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

=>

✅ Part 1: Document Requests in Rails

[4 Marks]

Definition:
In Ruby on Rails, **document requests** refer to client (browser or API) requests for specific resources from the server, such as HTML, JSON, XML, or JavaScript responses. Rails handles these requests using **MIME types** and **respond_to blocks** in controllers.

1. Request Format Handling

Rails determines the type of document requested by checking the `Accept` header or file extension in the URL.

```
ruby
CopyEdit
def show
  @user = User.find(params[:id])
  respond_to do |format|
    format.html # renders show.html.erb
    format.json { render json: @user }
    format.xml  { render xml: @user }
  end
end
```

2. Types of Document Responses:

Format	Description
`.html`	Standard web pages
`.json`	API response for JavaScript apps
`.xml`	Used in older APIs
`.js`	JavaScript responses (for AJAX)

3. Advantages:

- Supports multiple clients (browsers, mobile apps, APIs).
- Improves modularity and separation of concerns.
- Enables building RESTful APIs and AJAX-based apps.

✅ Part 2: Processing Forms in Rails

[4 Marks]

Rails provides powerful tools to handle HTML form submission and process the data securely and efficiently.

1. Creating a Form:

Use `form_with` or `form_for` helper to create forms.

```
erb
CopyEdit
<%= form_with model: @user, local: true do |form| %>
  <%= form.text_field :name %>
  <%= form.email_field :email %>
  <%= form.submit "Register" %>
<% end %>
```

- `@user` is an instance of a model (like `User.new`)
- `local: true` ensures a regular (non-AJAX) form submission.

---

2. Receiving Form Data in Controller:

```
ruby
CopyEdit
def create
  @user = User.new(user_params)
  if @user.save
    redirect_to @user, notice: "User created!"
```

```
  else
    render :new
  end
end

private

def user_params
  params.require(:user).permit(:name, :email)
end
```

- `params` holds all form data.
- `require` and `permit` ensure strong parameters for security.

---

3. Flow of Form Processing:

1. User fills out the form and submits.
2. HTTP POST request is sent to controller.
3. Controller reads form data via `params`.
4. Data is used to create or update model objects.
5. Based on success, redirect or re-render form.

✅ Conclusion:

- **Document requests** in Rails allow responding with various content types like HTML, JSON, or XML based on client needs.
- **Form processing** in Rails is secure and follows MVC structure, enabling developers to collect, validate, and store user input easily using built-in helpers and strong parameters.

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

=>

✅ Part 1: Classes in Ruby

[5 Marks]

◆ What is a Class?

A **class** in Ruby is a blueprint for creating objects. It defines **attributes** (variables) and **methods** (functions) that describe the behavior of an object.

◆ Defining a Class

```
ruby
CopyEdit
class Student
  def initialize(name, age)
    @name = name     # instance variable
    @age = age
  end

  def display
    puts "Name: #{@name}, Age: #{@age}"
  end
end
```

- `initialize` is a constructor method that runs when an object is created.
- `@name` and `@age` are **instance variables**.
- `display` is an **instance method**.

◆ Creating an Object from Class

```
ruby
CopyEdit
s1 = Student.new("John", 22)
s1.display    # Output: Name: John, Age: 22
```

◆ Features of Classes in Ruby

- Fully Object-Oriented (everything is an object)
- Supports **inheritance**, **encapsulation**, and **polymorphism**
- Allows defining **class methods** and **instance methods**

✅ Part 2: Arrays in Ruby

[4 Marks]

◆ What is an Array?

An **array** is an ordered collection of elements that can store values of any data type (integer, string, object, etc.).

◆ Creating Arrays

```
ruby
CopyEdit
arr = [10, "hello", 3.14]
```

◆ Accessing Elements

```
ruby
CopyEdit
puts arr[0]    # 10
```

```
puts arr[1]    # "hello"
```

◆ Common Array Methods

```
ruby
CopyEdit
arr.push("Ruby")        # Add element
arr.pop                 # Remove last element
arr.length              # Number of elements
arr.each { |x| puts x } # Iterate through array
```

◆ Multidimensional Array

```
ruby
CopyEdit
matrix = [[1, 2], [3, 4]]
puts matrix[1][1]    # Output: 4
```

◆ Arrays with Objects

You can also store objects inside arrays:

```
ruby
CopyEdit
s1 = Student.new("Amit", 20)
s2 = Student.new("Pooja", 21)
students = [s1, s2]

students.each { |s| s.display }
```

✅ Conclusion:

- **Classes** in Ruby define the structure and behavior of objects.
- **Arrays** are flexible data structures for storing and manipulating ordered collections.
- Together, they support Ruby's **object-oriented and dynamic programming model**, making it powerful for both small scripts and large applications.

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

=>

✅ Part 1: Rails with AJAX

[4 Marks]

◆ What is AJAX?

AJAX (Asynchronous JavaScript and XML) is a technique that allows web pages to **send and receive data from the server asynchronously** without reloading the whole page. This results in a smoother and faster user experience.

◆ **How Rails Supports AJAX:**

Ruby on Rails provides **built-in support for AJAX** using UJS (Unobtrusive JavaScript) and allows you to create dynamic, responsive interfaces.

◆ **Basic Flow of AJAX in Rails:**

1. User clicks a button or submits a form.
2. JavaScript sends an **AJAX request** to the server.
3. The server processes it and responds with JavaScript (JS), JSON, or HTML.
4. JavaScript updates the page content dynamically.

◆ **Example in Rails:**

View:

```
erb
CopyEdit
<%= button_to "Like", like_post_path(@post), remote: true %>
```

Controller:

```
ruby
CopyEdit
def like
  @post.increment!(:likes)
  respond_to do |format|
    format.js   # renders like.js.erb
  end
end
```

like.js.erb:

```
js
CopyEdit
alert("Post liked!");
```

◆ **Advantages of Rails with AJAX:**

- Faster user interaction (no full-page reload)
- Clean, responsive user interface
- Easy integration with Rails UJS

✅ Part 2: EJB (Enterprise JavaBeans)

[4 Marks]

◆ What is EJB?

EJB (Enterprise JavaBeans) is a **server-side software component** used in Java EE (Enterprise Edition) to build **scalable, secure, and transactional enterprise-level applications**.

◆ Types of EJB:

1. **Session Beans:**
   - Perform business logic.
   - Types: Stateless, Stateful, Singleton.
2. **Entity Beans:**
   - Represent persistent data stored in a database (now replaced by JPA).
3. **Message-Driven Beans:**
   - Handle asynchronous messages from a queue (JMS).

◆ Example: Stateless Session Bean

```
java
CopyEdit
@Stateless
public class CalculatorBean {
  public int add(int a, int b) {
    return a + b;
  }
}
```

◆ Key Features of EJB:

- **Remote Access:** Components can be accessed remotely.
- **Security:** Built-in support for role-based access control.
- **Transactions:** Automatic transaction management.
- **Scalability:** Ideal for large enterprise applications.

◆ Advantages of EJB:

- Reduces boilerplate code for transactions and security.
- Provides a standard way to create scalable business components.
- Allows distributed and asynchronous processing.

✅ Conclusion:

- **Rails with AJAX** enhances user experience by allowing asynchronous interactions without full page reloads, using Rails' built-in UJS and controller support.
- **EJB** provides a robust Java framework for building scalable, secure, and transactional enterprise applications using session and message-driven beans.

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

=>

✅ What is a Multiple Selection Construct?

A **multiple selection construct** allows a program to make decisions between **two or more paths** based on different conditions. In Ruby, this is mainly done using:

1. `case-when` statement
2. `if-elsif-else` ladder

✅ 1. `case-when` Statement (Like switch-case)

Ruby provides the `case-when` construct to handle multiple conditions more cleanly than `if-elsif`.

◆ Syntax:

```
ruby
CopyEdit
case expression
when value1
  # code to execute if expression == value1
when value2
  # code to execute if expression == value2
else
  # code to execute if no match
end
```

◆ Example:

```
ruby
CopyEdit
grade = "B"

case grade
when "A"
  puts "Excellent"
when "B"
  puts "Good"
```

```
when "C"
  puts "Average"
else
  puts "Fail"
end
```

Output: `Good`

◆ **Features:**

- Clean and readable for multiple comparisons.
- Can match ranges, regular expressions, and conditions.

✅ 2. `if-elsif-else` Ladder

Used when conditions are not just comparisons but include logical expressions or calculations.

◆ **Example:**

```
ruby
CopyEdit
marks = 78

if marks >= 90
  puts "Grade A"
elsif marks >= 75
  puts "Grade B"
elsif marks >= 60
  puts "Grade C"
else
  puts "Fail"
end
```

Output: `Grade B`

◆ **Features:**

- More flexible than `case` as it can use conditions like `>`, `<`, logical operators.
- Good for range-based or computed conditions.

✅ 3. Using `case` with `when` Ranges

You can also use **range values** in `when` for cleaner code.

```
ruby
CopyEdit
```

```
score = 85

case score
when 90..100
  puts "Outstanding"
when 75..89
  puts "Very Good"
when 60..74
  puts "Good"
else
  puts "Needs Improvement"
end
```

Output: `Very Good`

✅ Conclusion:

- Ruby supports **multiple selection constructs** using `case-when` and `if-elsif-else` statements.
- `case-when` is best for equality or range comparisons.
- `if-elsif` is more flexible when conditions are complex.
- Both allow writing **clear and efficient decision-making logic** in Ruby programs.

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

=>

✅ What is AJAX?

AJAX (Asynchronous JavaScript and XML) is a web technology that allows a web page to communicate with the server **without reloading the entire page**. This results in a faster, smoother, and more dynamic user experience.

✅ How Rails Works with AJAX

Ruby on Rails integrates AJAX functionality using its **Unobtrusive JavaScript (UJS)** mechanism. It enables developers to write less JavaScript manually and still implement powerful asynchronous features.

✅ Key Concepts:

1. Remote Requests in Views

You can enable AJAX on forms or buttons by setting `remote: true`.

```
erb
CopyEdit
<%= form_with url: likes_path, method: :post, remote: true do |form| %>
  <%= form.submit "Like" %>
<% end %>
```

This form will now **submit data via AJAX**, not as a full HTTP request.

---

2. Controller Action

```
ruby
CopyEdit
def create
  @like = Like.create(post_id: params[:post_id])
  respond_to do |format|
    format.js   # Looks for create.js.erb
  end
end
```

The `format.js` tells Rails to respond with JavaScript.

---

3. JavaScript Response (create.js.erb)

```
js
CopyEdit
alert("Post liked!");
// You can also use jQuery or JS to update parts of the page
$("#like-count").html("<%= @post.likes.count %>");
```

This file contains JavaScript that is run **on the client-side** when the AJAX request returns.

✅ Advantages of Using AJAX in Rails:

Benefit	Description
Faster Updates	Only part of the page updates, not the whole page.
Better UX	Smoother user experience and interactive pages.
Less Server Load	Fewer full-page reloads reduce server strain.
Easy to Use	Rails UJS makes AJAX implementation simple.

✅ Conclusion:

Rails and AJAX work together seamlessly using `remote: true` attributes and `respond_to` blocks. This enables developers to build **interactive, modern web applications** without writing heavy JavaScript code, improving performance and user experience.

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

=>

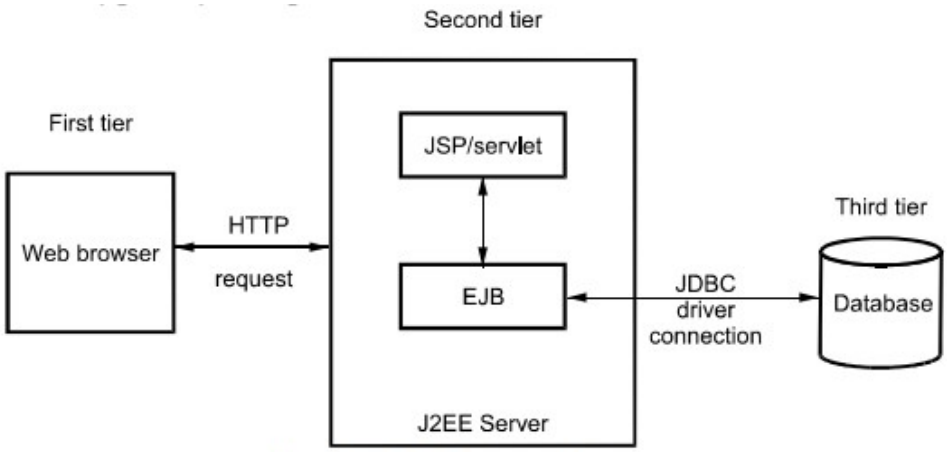


Fig. 6.17.3 3 Tier architecture

✅ What is an EJB Container?

The **EJB (Enterprise JavaBeans) container** is a part of the **Java EE (Enterprise Edition)** application server that provides a **runtime environment** for executing enterprise beans (EJBs). It manages the **lifecycle, security, transactions, concurrency, and resource pooling** of the EJB components.

✅ Diagram: EJB Container Architecture

```
pgsql
CopyEdit
+-----------------------------------------------------+
|                Application Server (JEE)             |
| +-------------------------------------------------+ |
| |                EJB Container                    | |
| |  +------------+   +-------------+   +--------+| |
| |  | Lifecycle  |   | Transaction |   | Security|| |
| |  | Management |   | Management  |   | Manager || |
| |  +------------+   +-------------+   +--------+| |
| |         |                 |              |   | |
| |     [Enterprise Beans (EJBs): Business Logic] | |
| +-------------------------------------------------+ |
+-----------------------------------------------------+
```

✅ Role of EJB Container in Enterprise Applications

Feature	Description
1. **Lifecycle Management**	Automatically creates, activates, passivates, and destroys EJB objects.
2. **Security Management**	Enforces **role-based access control** and secure communication.
3. **Transaction Management**	Supports **automatic transaction handling** (ACID properties) using JTA.
4. **Concurrency Control**	Ensures thread safety by handling **concurrent access** to session and entity beans.
5. **Resource Pooling**	Manages pooling of beans, database connections, and resources for efficiency.
6. **Remote Access Support**	Allows beans to be accessed remotely via **RMI or web services.**

✅ Types of Beans Managed by EJB Container

1. **Session Beans** – Handle business logic (Stateless, Stateful, Singleton)
2. **Message-Driven Beans** – Handle asynchronous messages from a queue
3. **Entity Beans** (now replaced by JPA) – Represent persistent data

---

✅ Conclusion

The EJB container acts as the **middleware layer** that simplifies development by handling complex enterprise-level services like transactions, security, and resource management. It allows developers to focus purely on **business logic** while the container takes care of the **infrastructure-level services**.

Q13. Explain how to write the methods and call the method in RUBY with example. [7]

=>

✅ What is a Method in Ruby?

A **method** is a set of expressions or code grouped together to perform a specific task. Methods help in **code reusability** and make programs organized and modular.

✅ How to Define a Method in Ruby?

- Use the `def` keyword followed by the method name.
- Method body contains the statements.
- End the method with `end`.
- Optionally, methods can take parameters.
- Methods return the last evaluated expression automatically.

Syntax:

```
ruby
CopyEdit

def method_name(parameters)
  # method body
end
```

✅ Example 1: Method Without Parameters

```
ruby
CopyEdit

def greet
  puts "Hello, Welcome to Ruby!"
end
```

Calling the method:

```
ruby
CopyEdit

greet
```

Output:

```
css
CopyEdit
Hello, Welcome to Ruby!
```

✅ Example 2: Method With Parameters and Return Value

```
ruby
CopyEdit

def add_numbers(a, b)
```

```
  sum = a + b
  return sum
end
```

OR since Ruby returns last expression by default, `return` is optional:

```
ruby
CopyEdit

def add_numbers(a, b)
  a + b
end
```

Calling the method:

```
ruby
CopyEdit

result = add_numbers(5, 3)
puts "Sum is #{result}"
```

Output:

```
csharp
CopyEdit

Sum is 8
```

✅ Additional Notes:

- **Methods can have default parameter values:**

```
ruby
CopyEdit

def greet(name = "Guest")
  puts "Hello, #{name}!"
end

greet          # Hello, Guest!
greet("Alice") # Hello, Alice!
```

- **Methods can take variable number of arguments using splat operator `*`:**

```
ruby
CopyEdit


def sum_all(*numbers)
  numbers.sum
end

puts sum_all(1,2,3,4)  # Output: 10
```

✅ Summary:

Step	Description
1. Define method	Use `def method_name(parameters)`
2. Write body	Add code to perform the task
3. End method	Use `end` to close method
4. Call method	Use `method_name(arguments)`
5. Use return	Optional; Ruby returns last expression by default

Q14. What are the difference between java beans and EJB? [5]

=>

Q14. What are the Differences Between JavaBeans and EJB? [5 Marks]

Aspect	JavaBeans	EJB (Enterprise JavaBeans)
Definition	JavaBeans are reusable software components written in Java, mainly used to encapsulate many objects into a single object (bean).	EJBs are server-side components used to build scalable, distributed, and transactional enterprise applications.
Purpose	Primarily designed for **component reuse** and to provide getter/setter methods to manage properties, often in GUI or simple applications.	Designed to handle **business logic**, transactions, security, and remote access in large enterprise applications.
Execution Environment	Runs on the client machine or simple Java applications, no special container required.	Runs inside an **EJB Container** on a Java EE server, which manages lifecycle, security, transactions, etc.
Services Provided	Provides no built-in services like transactions or security.	Provides automatic **transaction management**, **security**, **concurrency control**, and **remote method invocation**.
Complexity	Simple components with getter/setter methods.	Complex, supports distributed computing, supports different

Aspect	JavaBeans	EJB (Enterprise JavaBeans)
		types of beans (session, entity, message-driven).
Use Cases	GUI components, simple reusable parts, property management.	Enterprise applications needing scalability, transactions, security, and distributed access.

Summary:

| JavaBeans is for creating simple reusable components mainly for client-side and GUI applications. | EJB is a robust server-side component model for enterprise applications with built-in services like transactions and security. |

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

=>

✅ What is Ruby?

Ruby is a high-level, interpreted, general-purpose programming language created by **Yukihiro Matsumoto** in the mid-1990s. It is designed to be simple, productive, and easy to read with a focus on **developer happiness**.

Ruby supports **object-oriented, procedural, and functional programming** paradigms. It is widely used for web development, especially with the Ruby on Rails framework.

✅ Features of Ruby:

1. **Pure Object-Oriented Language**
   Everything in Ruby is an object, including numbers, strings, and even classes themselves.
2. **Simple and Readable Syntax**
   Ruby syntax is clean and resembles natural language, making it easy to learn and write.
3. **Dynamic Typing and Duck Typing**
   Ruby is dynamically typed, so variables don't need explicit types. Duck typing

allows flexible code where an object's methods and properties determine its usability.

4. **Automatic Memory Management**
   Ruby uses garbage collection to automatically free unused memory.
5. **Rich Standard Library**
   Ruby has a large standard library with built-in functions for many tasks like file handling, networking, and threading.
6. **Support for Mixins**
   Ruby uses modules to share reusable code across classes through mixins instead of multiple inheritance.
7. **Exception Handling**
   Ruby provides easy-to-use exception handling for robust error management.
8. **Highly Portable**
   Ruby runs on many platforms such as Windows, macOS, Linux, and Unix.
9. **Metaprogramming Capabilities**
   Ruby supports metaprogramming, enabling programs to write code dynamically.

Summary:

Ruby is a **flexible, easy-to-read, and powerful programming language** that supports multiple programming styles and is especially popular for web development.

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

=>

✅ What are Scalar Types in Ruby?

Scalar types are the basic, simple data types that store a single value. In Ruby, scalar types represent **primitive values** such as numbers, characters, and boolean values.

The most common scalar types in Ruby are:

- **Integer** — whole numbers
- **Float** — numbers with decimal points
- **String** — sequence of characters
- **Symbol** — immutable, interned string used as identifiers
- **Boolean** — `true` or `false`
- **NilClass** — represents `nil` (absence of value)

✅ 1. Integer

- Represents whole numbers (positive, negative, or zero).
- Ruby integers have unlimited size (no fixed limit).

Operations on Integers:

```
ruby
CopyEdit

a = 10
b = 3

puts a + b    # Addition: 13
puts a - b    # Subtraction: 7
puts a * b    # Multiplication: 30
puts a / b    # Division: 3 (integer division)
puts a % b    # Modulus: 1 (remainder)
puts a ** b   # Exponentiation: 1000 (10^3)
```

✅ 2. Float

- Represents real numbers with decimal points.

Operations on Floats:

```
ruby
CopyEdit

x = 10.5
y = 3.2

puts x + y      # Addition: 13.7
puts x - y      # Subtraction: 7.3
puts x * y      # Multiplication: 33.6
puts x / y      # Division: 3.28125
```

✅ 3. String

- Represents a sequence of characters enclosed in single or double quotes.

Common String Operations:

```
ruby
CopyEdit

str1 = "Hello"
str2 = "World"

puts str1 + " " + str2      # Concatenation: "Hello World"
puts str1 * 3               # Repetition: "HelloHelloHello"
puts str1.length            # Length of string: 5
```

```
puts str1.downcase          # Lowercase: "hello"
puts str1.upcase            # Uppercase: "HELLO"
puts str1.include?("ll")    # Check substring: true
```

✅ 4. Symbol

- Immutable, memory-efficient identifiers often used as keys in hashes or constants.
- Represented with a colon `:` before the name.

Example:

```
ruby
CopyEdit

:name
:age

# Usage as keys
person = { name: "Alice", age: 25 }
puts person[:name]   # Output: Alice
```

Symbols are faster than strings for repeated use because they are only stored once.

✅ 5. Boolean

- Ruby has two boolean values: `true` and `false`.

Boolean Operations:

```
ruby
CopyEdit

puts true && false    # Logical AND: false
puts true || false    # Logical OR: true
puts !true            # Logical NOT: false
```

✅ 6. NilClass (nil)

- Represents “nothing” or “no value”.

```
ruby
CopyEdit

value = nil
puts value.nil?      # Checks if value is nil: true
```

✅ Summary Table of Scalar Types

Scalar Type	Description	Example
Integer	Whole numbers	10, -5, 0
Float	Decimal numbers	3.14, -0.5
String	Text	"Hello"
Symbol	Immutable identifier	:name
Boolean	True or False	true, false
NilClass	No value (nil)	nil

Conclusion:

Ruby's scalar types provide the fundamental building blocks for data manipulation. They support a variety of operations such as arithmetic for numbers, string manipulation, boolean logic, and identity using symbols. Understanding these types and their operations is essential for effective Ruby programming.

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

=>

✅ Introduction to Rails

Ruby on Rails (Rails) is a powerful web application framework written in Ruby. It follows the **MVC (Model-View-Controller)** architecture and is designed to make web development faster and easier by following **convention over configuration** and **don't repeat yourself (DRY)** principles.

✅ Positive Aspects (Advantages) of Rails

Aspect	Explanation	Example/Note
1. **Convention over Configuration**	Rails provides default structures and conventions, reducing the need for explicit configuration.	Developers don't need to write XML config files; Rails auto-

Aspect	Explanation	Example/Note
		loads database tables with matching names.
2. **Rapid Development**	Provides many built-in tools and generators to speed up coding.	`rails generate scaffold Post title:string body:text` creates full CRUD code automatically.
3. **MVC Architecture**	Separates application logic, UI, and data, making the app organized and maintainable.	Models handle data, Views handle presentation, Controllers handle business logic.
4. **Rich Library and Gems**	Huge collection of gems (libraries) that add features like authentication, payments, and APIs.	Using Devise gem for user authentication simplifies adding login/signup.
5. **DRY Principle**	Avoids duplication by allowing reuse of code and shared templates/partials.	Shared header/footer partials in views prevent repeating code.
6. **Active Record ORM**	Simplifies database interactions by mapping database tables to Ruby classes.	Easily perform `Post.find(1)` instead of writing SQL queries.
7. **Built-in Testing Framework**	Supports automated unit, functional, and integration testing to ensure code quality.	Use RSpec or MiniTest for test-driven development (TDD).
8. **Strong Community Support**	Large community, regular updates, and many tutorials/resources available.	Frequent updates and many open-source plugins.
9. **RESTful Application Design**	Encourages use of REST architecture, making URLs and API design simple and meaningful.	`GET /posts/1` fetches a post; `POST /posts` creates a post.

✅ Example Demonstrating Rails Advantages

Suppose you want to create a **blog application** where users can create, read, update, and delete posts.

1. **Rapid scaffolding:**

```
bash
CopyEdit

rails generate scaffold Post title:string content:text
rails db:migrate
```

This command generates:

- Model (`Post`) for database interaction
- Controller for business logic (CRUD)
- Views for user interface
- Routes following RESTful conventions

2. **Active Record simplifies DB operations:**

```
ruby
CopyEdit

post = Post.new(title: "Hello Rails", content: "Rails is awesome!")
post.save
```

3. **Built-in security and session management:**

Add authentication easily using gems like **Devise**.

✅ Summary

Advantage	Impact
Convention over configuration	Faster coding, less boilerplate
Rapid Development	Quick prototyping and iteration
MVC Architecture	Clear separation of concerns
Rich Gems Ecosystem	Add complex features easily
DRY Principle	Cleaner, reusable code
Active Record ORM	Easy database interaction
Testing Framework	Reliable, maintainable code
Community Support	Abundant learning resources
RESTful Design	Clean URL and API structure

Conclusion:

Ruby on Rails is ideal for building modern web applications quickly and efficiently. Its conventions, powerful tools, and strong community support make it a favorite framework among developers.

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

=>

i) Rails with AJAX

- **AJAX** (Asynchronous JavaScript and XML) is a technique for creating fast, dynamic web pages by updating parts of a page without reloading the whole page.
- In **Ruby on Rails**, AJAX is integrated to improve user experience by making web applications more responsive and interactive.
- Rails provides built-in helpers and support for AJAX via:
  - `remote: true` option in forms or links to submit requests asynchronously.
  - JavaScript response templates (`.js.erb`) to update page content dynamically.
- Rails uses **Unobtrusive JavaScript (UJS)** to separate JavaScript code from HTML, maintaining clean code.
- Example:
  When submitting a form with `remote: true`, Rails sends an AJAX request. The server responds with JavaScript to update a section of the page without full reload, e.g., adding a comment dynamically.
- Benefits:
  - Improved performance and user experience.
  - Reduced server load and bandwidth usage.
  - Seamless dynamic updates on web pages.

---

ii) WAP and WML

- **WAP (Wireless Application Protocol):**
  A communication protocol designed to enable mobile devices like phones and PDAs to access internet content and services over wireless networks. It allows web-like browsing on small screens with limited bandwidth.
- **Components of WAP:**
  - WAP Gateway: Converts WAP requests into HTTP and vice versa.
  - WAP Browser: Mobile browser supporting WAP protocols.

    - WML: Wireless Markup Language, a markup language similar to HTML but optimized for small screens and limited bandwidth.
- **WML (Wireless Markup Language):**
    - XML-based markup language used to create pages for WAP devices.
    - Designed for low bandwidth and small display devices.
    - Supports cards (like pages) grouped in decks for navigation.
    - Elements include `<wml>`, `<card>`, `<p>`, `<do>` (for actions), `<input>`, etc.
- **Usage:**
  Early mobile web content was developed using WAP and WML to deliver simplified content to mobile devices before smartphones became common.

Summary Table

Topic	Description
Rails with AJAX	Technique to update web pages asynchronously without full reload, improving responsiveness in Rails.
WAP (Wireless Application Protocol)	Protocol for delivering web content to mobile devices over wireless networks.
WML (Wireless Markup Language)	Lightweight markup language for WAP devices, optimized for small screens and low bandwidth.

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

=>

6.17.1 Types of EJB

- Following are the types of EJB.

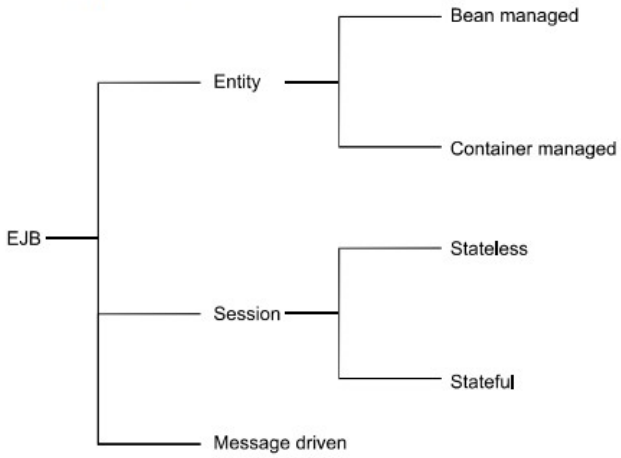


Fig. 6.17.1 Types of EJB

✅ What is EJB?

EJB (Enterprise JavaBeans) is a server-side component architecture for Java EE (Enterprise Edition) used to develop scalable, distributed, transactional, and secure enterprise-level applications.

- EJB simplifies development by handling system-level services like **transactions**, **security**, **remote access**, and **concurrency** automatically.
- It enables developers to focus on business logic rather than infrastructure concerns.
- EJB components run inside an **EJB container** (part of an application server) which manages their lifecycle and provides services.

✅ Types of EJBs

There are **three main types** of Enterprise JavaBeans, each serving different purposes:

---

1. Session Beans

- Represent a client's interaction with the application and contain business logic.
- They can be **stateful** or **stateless**.

Type	Description
Stateless Session Beans	Do not maintain any client-specific state between method calls. Suitable for lightweight operations.
Stateful Session Beans	Maintain state across multiple method calls for a specific client. Useful for conversational tasks (e.g., shopping carts).

Example:
A stateless session bean for a calculator service or a stateful session bean maintaining user session data.

---

2. Entity Beans

- Represent persistent data stored in a database.
- They act as an object-oriented interface to database records.
- Allow developers to manipulate database data as Java objects.

Note: In newer Java EE versions, **JPA (Java Persistence API)** has largely replaced entity beans.

Example:
An entity bean representing a `Customer` with attributes like `name`, `address`, etc.

---

3. Message-Driven Beans (MDBs)

- Handle asynchronous communication by listening to messages from a queue or topic (Java Message Service - JMS).
- Do not maintain any client state.
- Useful for processing messages like email notifications, order processing, etc.

Example:
A message-driven bean that processes orders placed in an e-commerce system asynchronously.

✅ Summary Table

Type	Purpose	State Maintenance	Use Case
Stateless Session Bean	Business logic without client state	No	Simple services like calculations
Stateful Session Bean	Business logic with client state	Yes	Shopping cart, user sessions
Entity Bean	Persistent data representation	Yes (database)	Customer, product records
Message-Driven Bean	Asynchronous message processing	No	Email processing, order queue

---

Conclusion:

EJB is a powerful Java EE technology that provides a standardized way to build reliable and scalable enterprise applications. Its types—session beans, entity beans, and message-driven beans—cover different aspects of business logic, data persistence, and asynchronous processing.

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

=>

✅ What is a Class in Ruby?

- A **class** in Ruby is a blueprint or template for creating **objects** (instances).
- It defines the properties (called **attributes**) and behaviors (called **methods**) that the objects created from the class will have.
- Classes help to organize code by grouping related data and functions.

✅ What is an Object in Ruby?

- An **object** is an instance of a class.
- When a class is defined, no memory is allocated until an object of that class is created.
- Each object has its own copy of the attributes defined in the class.

✅ Defining a Class in Ruby

```
ruby
CopyEdit
class Person
  # Initialize method to set attributes when object is created
  def initialize(name, age)
    @name = name      # instance variable
    @age = age
  end

  # Method to display person details
  def display
    puts "Name: #{@name}, Age: #{@age}"
  end
end
```

- `initialize` is a special method called automatically when a new object is created.
- `@name` and `@age` are **instance variables** unique to each object.

✅ Creating Objects from a Class

```
ruby
CopyEdit
# Create objects of Person class
person1 = Person.new("Alice", 25)
person2 = Person.new("Bob", 30)

# Call methods on objects
person1.display   # Output: Name: Alice, Age: 25
person2.display   # Output: Name: Bob, Age: 30
```

- `Person.new` creates a new object of the `Person` class.
- Each object stores its own data for `name` and `age`.

✅ Example: Adding Methods to Modify Object Data

```
ruby
CopyEdit
class Person
  def initialize(name, age)
    @name = name
    @age = age
  end

  def display
    puts "Name: #{@name}, Age: #{@age}"
  end

  # Setter method to update age
```

```ruby
  def set_age(new_age)
    @age = new_age
  end

  # Getter method to get name
  def get_name
    @name
  end
end

person = Person.new("Charlie", 28)
person.display        # Name: Charlie, Age: 28

person.set_age(29)
puts person.get_name  # Charlie
person.display        # Name: Charlie, Age: 29
```

✅ Key Concepts

Term	Explanation
Class	Blueprint to create objects
Object	Instance of a class
Instance Variable	Variable belonging to an object (`@name`)
Method	Function defined inside a class
`initialize` method	Special constructor method called when object is created

---

Conclusion:

- Classes and objects are fundamental to Ruby's object-oriented programming model.
- Classes define the structure and behavior, while objects are actual entities created from classes.
- Using classes and objects helps to write modular, reusable, and organized code.

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

=>

✅ Introduction to Rails Application

- **Ruby on Rails (Rails)** is a popular open-source **web application framework** written in Ruby.
- It follows the **Model-View-Controller (MVC)** architecture, which separates application logic into three interconnected parts:
  - **Model:** Manages data and business logic.
  - **View:** Handles the user interface and presentation.
  - **Controller:** Processes user requests and coordinates between Model and View.
- Rails emphasizes **convention over configuration**, **DRY (Don't Repeat Yourself)** principles, and rapid development.
- A typical Rails application is structured into folders like **app/models**, **app/views**, **app/controllers**, **config**, **public**, and others.

✅ Layouts in Rails

- **Layouts** are templates used to wrap views with a common structure (such as headers, footers, navigation menus).
- They help avoid repetition by allowing multiple views to share a common page design.
- By default, layouts are placed in the folder: `app/views/layouts/`.
- The default layout file is named **application.html.erb**, which wraps all views unless specified otherwise.

Example:

```
erb
CopyEdit
<!DOCTYPE html>
<html>
<head>
  <title>My Rails App</title>
  <%= csrf_meta_tags %>
  <%= csp_meta_tag %>
  <%= stylesheet_link_tag 'application', media: 'all' %>
</head>
<body>
  <header>
    <h1>Welcome to My App</h1>
  </header>

  <%= yield %> <!-- This renders the content of the specific view -->

  <footer>
    <p>© 2025 My Rails App</p>
  </footer>
</body>
</html>
```

- The `<%= yield %>` keyword is where the content from the view is inserted into the layout.

✅ Stylesheets in Rails

- Stylesheets control the **look and feel** of the application using CSS.
- Rails supports adding CSS files in the **app/assets/stylesheets/** directory.
- The main stylesheet is usually named `application.css` or `application.scss`.
- These stylesheets are linked automatically in the layout using the helper:

```
ruby
CopyEdit
<%= stylesheet_link_tag 'application', media: 'all' %>
```

- Rails uses the **Asset Pipeline** to manage and serve CSS files efficiently, allowing features like minification and concatenation.
- Developers can write CSS or use preprocessors like **Sass** to write modular and maintainable styles.

Summary:

Feature	Description
Rails Application	Web app framework based on MVC architecture and Ruby language
Layouts	Templates that wrap views with common page structure, avoid repetition
Stylesheets	CSS files controlling the visual design, managed via asset pipeline

---

Conclusion

Rails applications provide a structured, efficient way to build web apps with MVC architecture. Layouts help maintain a consistent page structure, and stylesheets control the visual appearance, both contributing to a maintainable and scalable application.

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

=>

✅ What are Scalar Types in Ruby?

- **Scalar types** are the basic data types that hold a single value.
- In Ruby, the common scalar types include:
  - **Integer** — whole numbers (e.g., 10, -5)
  - **Float** — decimal numbers (e.g., 3.14, -0.5)
  - **String** — sequences of characters (e.g., "Hello")
  - **Symbol** — lightweight identifiers used as names or keys (e.g., :name)
  - **Boolean** — true or false values (`true`, `false`)
  - **NilClass** — special type for `nil` (no value)

✅ Scalar Types with Examples

Type	Example	Description
Integer	`x = 42`	Numeric whole number
Float	`pi = 3.1415`	Decimal number
String	`name = "Ruby"`	Text data
Symbol	`:age`	Immutable, memory-efficient label
Boolean	`is_valid = true`	Logical true or false
NilClass	`value = nil`	Represents "nothing" or "no value"

✅ Operations on Scalar Types in Ruby

1. Integer and Float Operations:

- Ruby supports common arithmetic operations:

Operation	Example	Result
Addition	`5 + 3`	8
Subtraction	`10 - 4`	6
Multiplication	`6 * 7`	42
Division	`10 / 3`	3 (integer division)
Float Division	`10.0 / 3`	3.3333

Operation	Example	Result
Modulus	`10 % 3`	1
Exponent	`2 ** 3`	8

---

2. String Operations:

- Strings support concatenation, repetition, and interpolation:

Operation	Example	Result
Concatenation	`"Hello " + "World"`	`"Hello World"`
Repetition	`"Ruby" * 3`	`"RubyRubyRuby"`
Interpolation	`"Name: #{name}"`	Inserts variable value

---

3. Symbol Operations:

- Symbols are immutable and used mainly as identifiers or keys in hashes.
- They don't support arithmetic but are efficient for comparison.

---

4. Boolean Operations:

- Boolean values are used in conditional expressions with logical operators:

Operation	Example	Result
AND	`true && false`	false
OR	`true		
NOT	`!true`	false

---

5. NilClass

- `nil` represents the absence of value.
- It's often used for uninitialized variables or empty results.

Summary Table of Scalar Types and Operations

Scalar Type	Operations	Example
Integer	+, -, *, /, %, **	`5 + 3 = 8`
Float	Same as Integer but with decimals	`3.14 * 2 = 6.28`
String	Concatenation (+), repetition (*)	`"Hi " + "There"`
Symbol	Used as identifiers, immutable	`:name`
Boolean	Logical operators: &&,	
NilClass	Represents 'no value'	`value = nil`

Conclusion:

- Scalar types form the foundation of data handling in Ruby.
- They support a variety of operations that enable arithmetic calculations, string manipulations, logical decisions, and identification.
- Understanding these helps in writing clear and efficient Ruby programs.

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

=>

✅ Architecture of EJB (Enterprise JavaBeans)

EJB is a **server-side component architecture** used to build scalable, secure, and transactional enterprise-level Java applications.

Main Components of EJB Architecture:

1. **EJB Container:**
   - A runtime environment that manages the execution of enterprise beans.
   - Provides system-level services such as transaction management, security, lifecycle management, concurrency, and remote access.
   - Runs inside an **application server** like JBoss, WebLogic, or GlassFish.
2. **Enterprise Beans:**
   - Business components encapsulating business logic.

- Managed by the EJB container.
- Can be invoked remotely by clients.

3. **Client:**
   - The user or application that accesses the EJB components.
   - Can be a web client, application client, or another EJB.
4. **EJB Server:**
   - The application server providing the EJB container and related services.

EJB Architecture Diagram (Simplified):

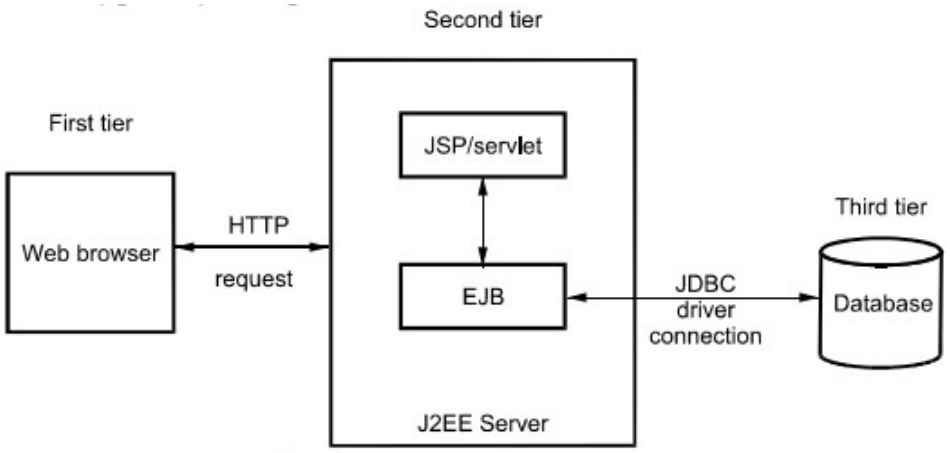


Fig. 6.17.3 3 Tier architecture

```
pgsql
CopyEdit
  +---------------------+
  |      Client         |
  +----------+----------+
             |
             v
  +---------------------+
  |    EJB Container    |
  |  - Transaction Mgmt |
  |  - Security         |
  |  - Lifecycle Mgmt   |
  |  - Persistence Mgmt |
  +----------+----------+
             |
             v
  +---------------------+
  |   Enterprise Beans  |
  | (Business Logic)    |
  +---------------------+
```

- The **client** calls the EJB container.
- The container manages the beans and provides services.

- Beans implement the actual business logic.

✅ Types of EJB

There are **three main types** of Enterprise JavaBeans:

---

1. Session Beans

- Represent a **single client's interaction** with the server.
- Can be **stateful** or **stateless**.

Types:

- **Stateless Session Bean:**
  - Does not maintain any client state between method calls.
  - Used for short-lived tasks like calculations or stateless services.
  - Example: A bean that processes payment but doesn't store user info.
- **Stateful Session Bean:**
  - Maintains state across multiple method calls or transactions for a particular client.
  - Useful when you need to keep user data temporarily.
  - Example: Shopping cart in an e-commerce app.
- **Singleton Session Bean:**
  - Only one instance per application.
  - Used for shared data or resources.
  - Example: Cache or configuration manager.

---

2. Entity Beans (Deprecated in EJB 3.0+)

- Represent persistent data stored in a database.
- Mapped to database tables.
- Managed either by the container or by the bean itself.
- Replaced by **Java Persistence API (JPA)** entities in modern applications.

---

3. Message-Driven Beans (MDB)

- Asynchronous message consumers.
- Process messages from **Java Message Service (JMS)** queues or topics.
- Do not maintain any state.
- Useful for loosely coupled, asynchronous processing.
- Example: Processing orders from a message queue.

Summary Table of EJB Types

Type	Description	Use Case
Stateless Session Bean	No client state maintained	Stateless services, calculations
Stateful Session Bean	Maintains client state between calls	Shopping carts, user sessions
Singleton Session Bean	Single instance, shared across application	Cache, configuration manager
Entity Bean	Persistent data representation (old style)	Database entity mapping (replaced by JPA)
Message-Driven Bean	Processes asynchronous JMS messages	Asynchronous processing, event handling

---

Conclusion

- EJB architecture separates business logic from system-level services via the **EJB container**.
- Different types of beans serve different purposes: session beans manage client interactions, message-driven beans handle asynchronous messages, and entity beans map persistent data.
- EJB simplifies enterprise application development by providing built-in services like transactions, security, and lifecycle management.