Web Technologies in Java HTML Tags: Anchor, Form, Table, Image, List Tags, Paragraph, Break, Label

Theory:

Q1)
Introduction to HTML and its structure.
Explanation of key tags:
o <a>: Anchor tag for hyperlinks.
o <form>: Form tag for user input.
o <table>: Table tag for data representation.
o <img>: Image tag for embedding images.
o List tags: <ul>, <ol>, and <li>.
o <p>: Paragraph tag.
o <br>: Line break.
o <label>: Label for form inputs.

Introduction to HTML and its Structure

HTML (HyperText Markup Language) is the standard language for creating web pages.

It defines the structure and layout of a webpage by using a series of elements (tags).

A basic HTML document structure:

```
<!DOCTYPE html>
<html>
<head>
   <title>My First Page</title>
</head>
<body>
   <h1>Hello, World!</h1>
   <p>This is a paragraph.</p>
</body>
</html>
```

Explanation:

<!DOCTYPE html> → Declares the document as HTML5.

<html> → Root element of the HTML document.

<head> → Contains metadata (title, styles, links, etc.).

<body> → Contains visible content (text, images, forms, tables, etc.).

Explanation of Key HTML Tags
1. <a>: Anchor Tag (Hyperlinks)

Used to create links to other webpages, files, or locations.

```
<a href="https://www.example.com">Visit Example</a>
```

href → Specifies the URL of the link.

2. <form>: Form Tag (User Input)

Used to collect user input (text, password, radio buttons, checkboxes, etc.).

```
<form action="submit.php" method="post">
   <label for="name">Name:</label>
   <input type="text" id="name" name="username">
   <input type="submit" value="Submit">
</form>
```

action → Where the data will be sent.

method → How data is sent (get or post).

3. <table>: Table Tag (Data Representation)

Displays data in rows and columns.

```
<table border="1">
   <tr>
      <th>Name</th>
      <th>Age</th>
   </tr>
   <tr>
      <td>Jayesh</td>
      <td>22</td>
   </tr>
</table>
```

<tr> → Table row.

<th> → Table header (bold).

<td> → Table data (cell).

4. <img>: Image Tag (Embedding Images)

Used to insert images in a webpage.

```
<img src="image.jpg" alt="Sample Image" width="200" height="150">
```

src → Path to the image.

alt → Alternate text if the image cannot load.

width & height → Dimensions.

5. List Tags (<ul>, <ol>, <li>)

Used to create lists.

```
<!-- Unordered List -->
<ul>
   <li>HTML</li>
   <li>CSS</li>
   <li>JavaScript</li>
</ul>

<!-- Ordered List -->
<ol>
   <li>First Step</li>
   <li>Second Step</li>
</ol>
```

<ul> → Unordered (bulleted) list.

<ol> → Ordered (numbered) list.

<li> → List item.

6. <p>: Paragraph Tag

Used to display paragraph text.

```
<p>This is a paragraph of text in HTML.</p>
```

7. <br>: Line Break

Inserts a line break without starting a new paragraph.

```
<p>Hello<br>World!</p>
```

8. <label>: Label for Form Inputs

Provides a label for input elements, improves accessibility.

```
<form>
   <label for="email">Email:</label>
   <input type="email" id="email" name="email">
</form>
```

Q2)
Theory:

Overview of CSS and its importance in web design.
Types of CSS:
o Inline CSS: Directly in HTML elements.
o Internal CSS: Inside a <style> tag in the head section.
o External CSS: Linked to an external file.

Overview of CSS and Its Importance in Web Design

CSS (Cascading Style Sheets) is a language used to control the presentation, layout, and design of web pages.

It was first introduced by the W3C (World Wide Web Consortium) in 1996 to separate content (HTML) from design (styling).

CSS allows developers to:

Change fonts, colors, spacing, and backgrounds.

Create responsive designs (websites that adapt to desktops, tablets, and mobiles).

Maintain consistency across multiple web pages with one stylesheet.

Reduce redundancy by avoiding repeating styles inside each HTML page.

Modern CSS also supports animations, transitions, grid layouts, flexbox, and media queries, making it essential for professional web design.

Types of CSS
1. Inline CSS

CSS is written directly inside an HTML element using the style attribute.

Useful for quick fixes or testing, but not recommended for large projects (because it mixes content with design).

```
<p style="color: blue; font-size: 18px;">This is a paragraph with inline CSS.</p>
```

2. Internal CSS

CSS is defined inside a <style> tag within the <head> section of an HTML file.

Good for single-page websites where the design is not reused across multiple pages.

```
<!DOCTYPE html>
<html>
<head>
   <style>
      body { background-color: lightyellow; }
      h1 { color: green; text-align: center; }
      p { font-size: 16px; color: darkblue; }
   </style>
</head>
<body>
   <h1>Welcome</h1>
   <p>This page uses internal CSS.</p>
</body>
</html>
```

3. External CSS

CSS is written in a separate .css file and linked to the HTML file using the <link> tag.

Best practice for large websites, as one file can control the styling of multiple pages.

```
<!-- index.html -->
<!DOCTYPE html>
```

```
<html>
<head>
   <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
   <h1>External CSS Example</h1>
   <p>This page uses an external CSS file.</p>
</body>
</html>

/* styles.css */
body {
   background-color: #f0f0f0;
}
h1 {
   color: crimson;
   text-align: center;
}
p {
   font-size: 18px;
   color: navy;
}
```

Q3)

Definition and difference between margin and padding.

Definition
Margin

The margin in CSS is the space outside the element’s border.

It creates distance between the element and surrounding elements.

It is used to control layout spacing.

Example:

```
div {
   margin: 20px;  /* Adds space outside the element */
}
```

Padding

The padding in CSS is the space inside the element’s border but around its content.

It creates space between the content and the border of the element.

It is used to control the inner spacing of an element.

Example:

```
div {
   padding: 20px;  /* Adds space inside the element */
}
```

Key Difference Between Margin and Padding

Feature	Margin (Outside)	Padding (Inside)
Definition	Space outside the border	Space inside the border, around content
Effect	Creates space between elements	Increases spacing between content and border
Background color	Does not affect background color (transparent area)	Background color fills padding area
Usage	Used to adjust external spacing (gap between elements)	Used to adjust internal spacing (space inside element)

Q4)
 How margins create space outside the element and padding creates space inside.

```
 _________________________________________
|           MARGIN                |  <- Margin (outside space)
|  ___________________________________   |
| |          BORDER             | |
| |  ________CONTENT_________      | |
| | |      PADDING        |    |  <- Padding (inside space)
| | |  Actual Text/Image    |    |
| | |__________________________|    |
| |___________________________________|  |
|_________________________________________|
```

Margin = Space outside the box

Padding = Space inside the box

Q5 Introduction to CSS pseudo-classes like :hover, :focus, :active, etc.

Introduction to CSS Pseudo-classes

A pseudo-class in CSS is used to define the special state of an element.
It allows you to apply styles to elements based on their interaction or position without needing extra classes or JavaScript.

Commonly Used Pseudo-classes

:hover

Applied when the user places the mouse pointer over an element.

Often used for links and buttons.

```
a:hover {
  color: red;
  text-decoration: underline;
}
```

:focus

Applied when an element (like an input field) gains focus, e.g., when clicked or navigated using the keyboard.

```
input:focus {
  border: 2px solid blue;
  background-color: #f0f8ff;
}
```

:active

Applied when an element is being clicked or activated.

Example: A button changes color when pressed.

```
button:active {
  background-color: green;
  transform: scale(0.95);
}
```

:visited

Styles links that the user has already visited.

```
a:visited {
  color: purple;
}
```

:first-child / :last-child

Selects the first or last child element inside a parent.

```
p:first-child {
  font-weight: bold;
}
p:last-child {
  color: blue;
}
```

Q6 Use of pseudo-classes to style elements based on their state.

Use of Pseudo-classes Based on Element State

Interactive states:
:hover, :focus, and :active help improve user experience by giving feedback when users interact with links, buttons, or form fields.

Navigation states:
:visited tells users where they’ve already been.

Structural states:
:first-child, :last-child, :nth-child() are useful for styling elements based on their position in the DOM.

Q7 Difference between id and class in CSS.

Feature	**id**	**class**
Definition	Used to uniquely identify a single element on a page.	Used to group multiple elements under the same style.
Selector Symbol	`#` (hash)	`.` (dot)
Uniqueness	Must be **unique** within a page. Only one element should have a particular `id`.	Can be **reused** across multiple elements.
Specificity	Higher specificity than class → overrides class styles if conflicts occur.	Lower specificity than `id`.
Usage	Best for targeting **one special element**.	Best for applying styles to a **group of elements**.

Q8Usage scenarios for id (unique) and class (reusable).

For repeated styles applied to multiple elements.

Example: Buttons, product cards, paragraphs with the same design.

```
<p class="highlight">This is important text.</p>
<p class="highlight">This is another highlighted text.</p>
```

```
.highlight {
  background-color: yellow;
  font-weight: bold;
}
```

Q8 Overview of client-server architecture.

Client-Server Architecture
Overview

Client-Server Architecture is a distributed computing model where tasks are divided between two main entities:

Client → the user’s device/application that requests services.

Server → a powerful system that provides resources, data, or services.

Communication happens over a network (e.g., Internet, LAN) using defined protocols like HTTP, HTTPS, FTP, SMTP, etc.

Example: When you open a website:

Your browser (client) sends a request →

The web server processes it and sends back the web page (response).

Q9 Difference between client-side and server-side processing.

| **Aspect** | **Client-side Processing** | **Server-side
Processing** |

Location	Runs on the **user’s device/browser**	Runs on the **server**
Technology	HTML, CSS, JavaScript	PHP, Java, Python, Node.js, .NET, Databases
Execution Speed	Fast (no network delay) but limited by device performance	Slower due to network round-trips, but more powerful processing
Security	Less secure (code visible in browser)	More secure (code hidden on server)
Examples	Form validation using JavaScript, dynamic styling, animations	User authentication, database queries, business logic
Dependency	Depends on user’s browser/device capability	Depends on server’s power and configuration

Q10 Roles of a client, server, and communication protocols.

Roles in Client-Server Model
(I) Client

Initiates communication by sending a request.

Usually lightweight (browser, mobile app, desktop app).

Displays results to the user.

(II) Server

Waits for requests, processes them, and returns responses.

Stores and manages resources, databases, applications, services.

Ensures security, authentication, and data integrity.

(III) Communication Protocols

Define how client and server exchange data reliably.

Examples:

HTTP/HTTPS → Web communication

FTP → File transfer

SMTP/IMAP/POP3 → Email services

TCP/IP → Basic internet communication

Q11)Introduction to the HTTP protocol and its role in web communication.

Introduction to HTTP Protocol

HTTP (Hypertext Transfer Protocol) is the foundation of web communication.

It is a stateless, application-layer protocol used for transmitting data (HTML, images, videos, JSON, etc.) between client (browser/app) and server.

Works on top of TCP/IP.

By default, uses port 80 (or 443 for secure HTTPS).

Role in Web Communication:

A client (browser) sends an HTTP request to a server.

The server processes it and sends back an HTTP response.

This request–response cycle enables everything we see on the web.

Q12) Explanation of HTTP request and response headers.

----------------HTTP Request---------------------------

When a browser asks for a web page, it sends an HTTP request.
It consists of:

Request line → Method + URL + Protocol version

Example: GET /index.html HTTP/1.1

Request headers → Metadata about the request (browser info, accepted formats, etc.)

Body (optional) → Data sent with the request (used in POST, PUT methods).

Common Request Headers

Host → Specifies the domain (e.g., Host: www.example.com)

User-Agent → Info about the browser/device

Accept → Data formats client can accept (e.g., text/html, application/json)

Content-Type → Type of data sent in request body (e.g., application/json)

Authorization → Credentials for authentication

---------------HTTP Response-----------------------------

After processing, the server sends an HTTP response.
It consists of:

Status line → Protocol + Status code + Status message

Example: HTTP/1.1 200 OK

Response headers → Metadata about the response (server details, caching rules, etc.)

Body → Actual data (HTML page, JSON, image, etc.)

Common Response Headers

Content-Type → Type of data returned (e.g., text/html, application/json)

Content-Length → Size of the response body

Server → Info about the web server (e.g., Apache, Nginx)

Set-Cookie → Sends cookies to the client for session management

Cache-Control → Defines caching policies

Q13) Introduction to J2EE and its multi-tier architecture.

Introduction to J2EE and Its Multi-Tier Architecture

-----------------What is J2EE?------------------------------

J2EE (Java 2 Platform, Enterprise Edition) is a platform for building distributed, enterprise-level applications using Java.

It extends Java SE with APIs for web, business logic, and enterprise services (like Servlets, JSP, EJB, JDBC, JMS, JPA, etc.).

The goal: To provide a scalable, secure, and portable environment for web and enterprise apps.

--------------------J2EE Multi-Tier Architecture-----------------------

J2EE applications are usually structured into multi-tiers (layers):

Client Tier (Presentation Layer)

Interacts with the end-user (browser, mobile app, desktop app).

Example: HTML, CSS, JavaScript, JSP pages.

Web Tier (Web Layer)

Handles client requests and responses.

Uses Servlets and JSP inside a Web Container.

Responsible for request routing, input validation, session handling.

Business Tier (Application Layer)

Contains the business logic of the application.

Implemented using EJB (Enterprise JavaBeans) or POJOs (Plain Old Java Objects).

Runs inside an Application Server.

Data Tier (Database Layer)

Responsible for storing and retrieving data.

Managed using Database Servers (e.g., MySQL, Oracle, PostgreSQL).

Accessed via JDBC, JPA, Hibernate.

Q14) Role of web containers, application servers, and database servers.

---------------Role of Different Components-----------------

(I) Web Container (Servlet/JSP Container)

Part of the Web Server that manages Servlets and JSPs.

Responsibilities:

Process client requests (HTTP).

Manage lifecycle of Servlets/JSPs.

Handle session management and security.

Example: Apache Tomcat, Jetty.

(II) Application Server

Provides a runtime environment for business logic (EJBs, JTA, JMS, etc.).

Responsibilities:

Execute complex business logic.

Transaction management.

Messaging and distributed services.

Example: JBoss/WildFly, GlassFish, WebLogic, WebSphere.

(III) Database Server

Stores persistent data of the application.

Responsibilities:

Execute SQL queries.

Ensure data consistency, security, and transactions.

Example: MySQL, Oracle, PostgreSQL, MS SQL Server.

Q15) Introduction to CGI (Common Gateway Interface).

Introduction to CGI (Common Gateway Interface)

CGI (Common Gateway Interface) is a standard protocol used to enable web servers to interact with external programs or scripts.

It allows a web server to pass user requests (form data, query parameters, etc.) to an external program (written in C, C++, Perl, Python, etc.) and then send the program's output back to the client's browser.

CGI was one of the earliest ways to create dynamic web pages before modern technologies like Servlets, PHP, and ASP.NET.

Q16 Process, advantages, and disadvantages of CGI programming.

---------------Advantages of CGI-------------------

Language Independent → Can be written in C, C++, Perl, Python, Shell Script, etc.

Simple to Implement → Easy to connect HTML forms with backend logic.

Portable → Works across different operating systems and web servers.

Good for Prototyping → Quick way to add dynamic behavior to web pages.

---------------Disadvantages of CGI------------------

Performance Issues

Each request spawns a new process → high memory and CPU usage.

Slow compared to persistent server-side solutions (like Servlets or PHP).

Scalability Problems

Not suitable for high-traffic websites because of process creation overhead.

Security Risks

Poorly written CGI scripts can lead to vulnerabilities (buffer overflows, code injection).

Obsolete for Modern Applications

Replaced by more efficient technologies (Servlets, JSP, PHP, Node.js, etc.).

Q17 Introduction to servlets and how they work.

A Servlet is a Java program that runs on a server and handles client requests (usually HTTP) and generates dynamic web content.

How Servlets Work (Lifecycle & Process)

Client Request

A browser sends an HTTP request (e.g., GET /login).

Web Container

The container (Tomcat, GlassFish, etc.) receives the request and checks for the mapped Servlet.

Servlet Execution

Loading & Initialization → init() method runs once when the Servlet is first loaded.

Request Handling → For every request, the container calls service() → which then calls doGet(), doPost(), etc., depending on request type.

Destruction → When the server shuts down, the destroy() method is called.

Response Generation

The Servlet processes the request (e.g., database query) and generates a response (HTML, JSON, XML, etc.).

Response Sent

Web container sends the response back to the client.

Q18 Advantages and disadvantages compared to other web technologies.

--------------------Advantages of Servlets-----------------------------

(I)Better Performance

Unlike CGI, Servlets don’t create a new process per request → more efficient.

(II)Portability

Written in Java → runs on any server with a compliant container.

(III)Integration with Java APIs

Can easily use JDBC, RMI, EJB, etc.

(IV)Security

Java’s security features and container-managed authentication make Servlets safer.

(V)Scalability

Can handle multiple requests efficiently using multithreading.

---------------------Disadvantages of Servlets------------------------------

(I)Complexity

Writing HTML code inside Java (Servlets) can become messy.

That’s why JSP (JavaServer Pages) was introduced to separate business logic from presentation.

(II)Verbosity

Compared to modern frameworks (Spring Boot, Node.js, Django), raw Servlets require more boilerplate code.

(III)Limited Modern Use

Direct Servlet programming is rare today. They are often used as the foundation for frameworks (e.g., Spring MVC, Struts) rather than directly.

Q19 History of servlet versions.

Version	**Year**	**Key Features**
Servlet 1.0	1997	First release by Sun Microsystems; basic API to handle requests/responses.
Servlet 2.1	1999	Introduced **RequestDispatcher** (forward & include), deployment descriptor (`web.xml`).
Servlet 2.2	1999	Introduced concept of **web applications** and packaging in **WAR files**.

Servlet 2.3	2001	Added **filter** and **listener** support; improved session handling.
Servlet 2.4	2003	Better XML support for `web.xml`.
Servlet 2.5	2005	Support for **annotations**, eliminating some need for `web.xml`.
Servlet 3.0	2009	Major update: **asynchronous support**, more annotations, pluggability (no need for full `web.xml`).
Servlet 3.1	2013	**Non-blocking I/O** added for better scalability.
Servlet 4.0	2017	Support for **HTTP/2** protocol.
Servlet 5.0	2020	Migration from **Java EE → Jakarta EE**, package renamed to `jakarta.servlet.*`.
Servlet 6.0	2022	Part of Jakarta EE 10; aligned with modern Java versions.

Q20 Types of servlets: Generic and HTTP servlets.

Types of Servlets

Servlets can be broadly divided into two types:

--------------------- Generic Servlet---------------------------------

An abstract class (javax.servlet.GenericServlet) that implements the Servlet interface.

Protocol-independent → can be used for any type of request (not just HTTP).

Must override the service() method.

Example: Generic Servlet

```
import java.io.*;
import javax.servlet.*;

public class MyGenericServlet extends GenericServlet {
   public void service(ServletRequest req, ServletResponse res)
          throws ServletException, IOException {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<h2>Hello from Generic Servlet</h2>");
   }
}
```

------------------ HTTP Servlet----------------------

A subclass of GenericServlet → specialized for HTTP protocol.

Found in javax.servlet.http.HttpServlet.

Provides methods like:

doGet() → handles GET requests

doPost() → handles POST requests

doPut(), doDelete(), etc.

Most commonly used in real-world web apps.

Example: HTTP Servlet

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyHttpServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<h2>Hello from HTTP Servlet (GET request)</h2>");
    }
}
```

Q21 Detailed comparison between HttpServlet and GenericServlet.

| **Aspect** | **GenericServlet** |
HttpServlet
|
| ---------------------- | -- |

--------------------- |
| **Inheritance** | Extends `java.lang.Object` and implements `Servlet`, `ServletConfig`, `Serializable` interfaces. | Extends **`GenericServlet`** and implements **HTTP-specific** features.

|
| **Protocol Support** | **Protocol-independent** → can be used for any type of request (FTP, SMTP, custom protocols). | **HTTP-specific** → designed to handle HTTP requests and responses.

|
| **Methods to Override** | Must **override `service()`** method to handle requests.
| Usually override **`doGet()`, `doPost()`, `doPut()`, `doDelete()`** depending on request type.

|

| **Ease of Use** | Less convenient → developer must handle request type manually inside `service()`. | Easier for HTTP → framework calls correct `doXxx()` method automatically. |

| **Common Usage** | Rarely used in practice (only for protocol-independent applications). | Widely used in **web applications** (JSP, MVC frameworks, REST APIs). |

| **Lifecycle Methods** | Inherits `init()`, `service()`, and `destroy()` from `Servlet`.
| Inherits everything from `GenericServlet` + adds HTTP methods (`doGet`, `doPost`, etc.).
|

| **Request & Response** | Works with `ServletRequest` and `ServletResponse` objects.
| Works with **`HttpServletRequest`** and **`HttpServletResponse`**, which provide extra HTTP features (headers, cookies, session handling, etc.). |

| **Flexibility** | More general, but requires more code for handling protocols.
| Specialized for web; simplifies handling of HTTP protocol.

|
| **Example Use Case** | A custom server handling **non-HTTP requests**.
| A **web application** handling HTTP requests (login, forms, APIs).

|

Q22 Explanation of the servlet life cycle: init(), service(), and destroy() methods.

A Servlet is managed by a Web Container (e.g., Tomcat).
The life cycle of a servlet defines how it is created, initialized, used, and destroyed.
It mainly involves three methods: init(), service(), and destroy().

(I) init() Method

Called by: Web container once when the servlet is first loaded.

Purpose: Initialize the servlet. Any one-time setup (like database connections, reading config) is done here.

Signature:

```
public void init() throws ServletException
```

(II)   service() Method

Called by: Web container for each client request.

Purpose: Handle requests and generate responses.

Signature (GenericServlet):

```
public void service(ServletRequest req, ServletResponse res) throws ServletException,
IOException
```

(III)
destroy() Method

Called by: Web container once before removing the servlet from memory (e.g., server shutdown).

Purpose: Cleanup resources like closing database connections or releasing memory.

Signature:

```
public void destroy()
```

Q23 How to create servlets and configure them using web.xml.

A Servlet is a Java class that extends HttpServlet and overrides methods like doGet() or doPost().

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<h2>Hello, Welcome to Servlets!</h2>");
   }
}
```

Steps explained:

Extend HttpServlet.

Override doGet() (for GET requests).

Set content type for response.

Use PrintWriter to write output.

Configuring Servlet Using web.xml

web.xml is the deployment descriptor for a web application.
It tells the Web Container how to load and map servlets.

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

    <!-- Servlet Declaration -->
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>

    <!-- Servlet Mapping -->
    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>

</web-app>
```

Q24)Explanation of logical URLs and their use in servlets.

Logical URL:

A logical URL (also called a virtual URL or alias) is a user-friendly name that does not directly represent the physical location of a resource (like a .class or .jsp file).

Instead, it is mapped to a servlet through the deployment descriptor (web.xml) or annotations (@WebServlet).

Uses of Logical URLs in Servlets:

=>   Abstraction: Users don’t need to know the actual servlet class name or package.

=>   Security: Direct access to servlet classes is prevented.

=> Flexibility: You can change the servlet implementation without changing the URL exposed to clients.

=> Maintainability: URLs remain consistent even if the servlet class changes.

Q25 Overview of ServletConfig and its methods.

Definition:

ServletConfig is an object provided by the servlet container to a servlet at initialization time.

It contains initialization parameters and a reference to the ServletContext.

Each servlet has its own ServletConfig object.

Methods of ServletConfig:

(I) String getInitParameter(String name)
→ Returns the value of the given initialization parameter.

(II) Enumeration<String> getInitParameterNames()
→ Returns the names of all initialization parameters.

(III) ServletContext getServletContext()
→ Returns a reference to the ServletContext (application-level object).

(IV) String getServletName()
→ Returns the logical name of the servlet (as given in web.xml).

Q26 Explanation of RequestDispatcher and the forward() and include() methods.

Definition:

RequestDispatcher is an interface provided by the servlet API to forward a request to another resource (servlet, JSP, or HTML) or to include the content of another resource in the response.

It allows server-side communication between resources without the client knowing about it.

Getting RequestDispatcher object

You can obtain it in two ways:

From ServletRequest (relative to current request path):

```
RequestDispatcher rd = request.getRequestDispatcher("page.jsp");
```

From ServletContext (absolute path from application root):

```
RequestDispatcher rd = getServletContext().getRequestDispatcher("/page.jsp");
```

Q27Introduction to ServletContext and its scope.

ServletContext

ServletContext is an interface that defines a communication channel between a web application and the servlet container (like Tomcat).

It is a shared memory space available to all servlets, JSPs, and filters in the same web application.

Scope	Lifetime	Shared with
Request	For a single HTTP request	Same request only

| **Session** | For one user session (until logout/timeout) | Same user |

| **ServletContext** | From application startup to shutdown | Across all users & servlets |

Q28 How to use web application listeners for lifecycle events.

Q29What are filters in Java and when are they needed?

Definition:

A filter in Java EE/Servlet API is an object that intercepts requests and responses before they reach a servlet (or after a servlet has processed them).

Implemented using the javax.servlet.Filter interface.

Purpose / Why Needed?
Filters are mainly used for pre-processing and post-processing of requests and responses.

Q30 Filter lifecycle and how to configure them in web.xml.

Like servlets, filters also have a defined lifecycle managed by the container:

init(FilterConfig config)

Called once when the filter is created.

Used for initialization (like reading filter configuration from web.xml).

doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

Called for each request passing through the filter.

You can:

Pre-process request

Call chain.doFilter(request, response) → pass request to next filter/servlet

Post-process response

destroy()

Called once when the filter is taken out of service.

Used for cleanup (close resources, release connections, etc.).

Q31Introduction to JSP and its key components: JSTL, custom tags, scriplets, and implicit objects.

JavaServer Pages (JSP) is a server-side technology that allows embedding Java code into HTML pages for building dynamic web applications.

JSP is translated into a Servlet by the web container (like Tomcat).

It simplifies web development by separating presentation (HTML) from business logic (Java code).

Key Components of JSP

JSTL (JSP Standard Tag Library)

JSTL provides a set of ready-made tags for common tasks, reducing the need for Java code in JSP.

Helps follow MVC by minimizing Java code inside JSP.
Tag Libraries:

Core Tags (c:if, c:forEach, c:choose, etc.) – for control flow.

Formatting Tags (fmt:formatNumber, fmt:formatDate) – for localization/formatting.

SQL Tags (sql:query, sql:update) – for database access (not recommended in production).

XML Tags – for working with XML.

Functions Tags (fn:length, fn:contains) – for string operations.