***Module 3. Web Technologies in Java***

1. **HTML Tags: Anchor, Form, Table, Image, List Tags, Paragraph, Break, Label**

**1.1- Introduction to HTML and its structure**

HTML is the standard markup language for creating Web pages.

HTML stands for Hyper Text Markup Language

HTML is the standard markup language for creating Web pages

HTML describes the structure of a Web page

HTML consists of a series of elements

HTML elements tell the browser how to display the content

HTML elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

* **Structure**

<!DOCTYPE html>  
<html>  
<head>  
<title>Page Title</title>  
</head>  
<body>

<h1>My First Heading</h1>  
<p>My first paragraph.</p>  
  
</body>  
</html>

1.2- **Explanation of key tags:**

**1. <a>: Anchor Tag (Hyperlink):-**

Used to create links to other pages or websites.

* **Example:**

<a href="https://www.example.com">Go to Example</a>

**2. <form>: Form Tag**

Used to collect user input, like text fields or buttons.

* **Example:**

<form action="submit.php">

<input type="text" name="username">

<input type="submit" value="Submit">

</form>

**3. <table>: Table Tag**

Used to display data in rows and columns.

* **Example:**

<table>

<tr>

<th>Name</th>

<th>Age</th>

</tr>

<tr>

<td>Utsav</td>

<td>25</td>

</tr>

</table>

**4. <img>: Image Tag**

Used to display an image on the page.

* **Example:**

<img src="image.jpg" alt="Description">

**5. List Tags: <ul>, <ol>, <li>**

Used to create unordered (bulleted) and ordered (numbered) lists.

* **Example (Unordered List):**

<ul>

<li>Apple</li>

<li>Banana</li>

</ul>

* **Example (Ordered List):**

<ol>

<li>First Item</li>

<li>Second Item</li>

</ol>

**6. <p>: Paragraph Tag**

Used to create a paragraph of text.

* **Example:**

<p>This is a paragraph.</p>

**7. <br>: Line Break**

Used to break the text onto a new line.

* **Example:**

<p>Line 1<br>Line 2</p>

**8. <label>: Label Tag**

Used to define a label for a form input field.

* **Example:**

<label for="name">Name:</label>

<input type="text" id="name">

These are the basic tags used to structure web content.

1. **CSS: Inline CSS, Internal CSS, External CSS**
   1. **Overview of CSS and its importance in web design.**

CSS stands for Cascading Style Sheets

CSS describes how HTML elements are to be displayed on screen, paper, or in other media

CSS saves a lot of work. It can control the layout of multiple web pages all at once

External stylesheets are stored in CSS files

### CSS Example :-

body {  
  background-color: lightblue;  
}  
  
h1 {  
  color: white;  
  text-align: center;  
}  
  
p {  
  font-family: verdana;  
  font-size: 20px;  
}

* **Importance in web design.**
  + - * CSS is important in web design because it controls how a website looks.
      * CSS helps make websites visually appealing by allowing you to change colors, fonts, layouts, and more.
      * With CSS, a website can be designed to adjust its layout depending on the screen size.
      * CSS also helps in maintaining consistency across all pages of a website.

**2.2 Types of CSS:**

* + **Inline CSS: Directly in HTML elements:-**

This type of CSS is written directly within an HTML element using the style attribute. It applies styles to a single element.

* **Example**:

<p style="color: blue; font-size: 18px;">

This is a blue paragraph.</p>

**Internal CSS**:

This CSS is placed inside the <style> tag within the <head> section of the HTML document. It styles the whole page but only affects the specific page it is written on.

* **Example**:

<head>

<style>

p { color: red; }

</style>

</head>

<body>

<p>This paragraph will be red.</p>

</body>

* + **External CSS**:

This type of CSS is written in a separate .css file and linked to the HTML document. It is used to style multiple pages consistently and makes the website easier to maintain.

* **Example**:

<link rel="stylesheet" href="styles.css">

1. **CSS: Margin and Padding** 
   1. - Definition and difference between margin and padding.

* **Margin**:

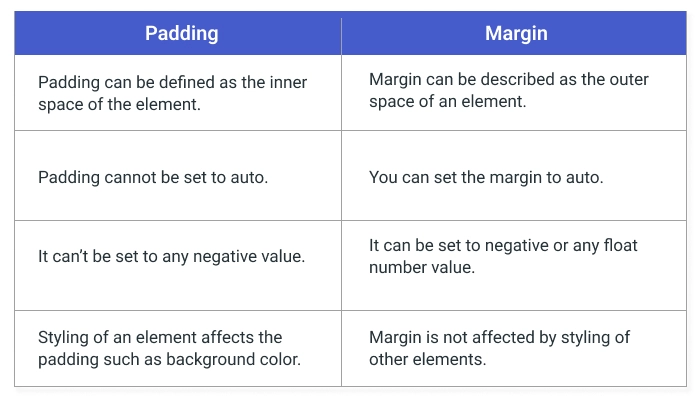
The margin is the space **outside** the border of an element. It creates space between the element and other surrounding elements on the page.

* **Padding**:

The padding is the space **inside** the border of an element. It creates space between the content of the element and its border.

**Difference between Margin and Padding:**

* **Margin** creates space **outside** the element, pushing other elements further away from it.
* **Padding** creates space **inside** the element, pushing the content (like text or images) away from the border.



**3.2- How margins create space outside the element and padding creates space inside:-**

Space Outside the Element

* Margins create space outside an element's border. They push the element away from other elements or the edge of the page, creating distance between them.
* When you set a margin, it increases the space around the element, moving it further from other nearby elements.

**Space Inside the Element**

* Padding creates space **inside** the element, between the content (like text or images) and the element's border. It pushes the content away from the border.
* When you set padding, it increases the space within the element, making the content smaller by pushing it inward.

|  |
| --- |
|  |

**4.** **CSS: Pseudo-Class**

* 1. **Introduction to CSS pseucdo-classes like :hover, :focus, :active, etc:-**

CSS pseudo-classes are used to define the special state of an element.

**1. :hover -**

* The :hover pseudo-class is used to style an element when the mouse pointer hovers over it.
* **Example**:

a:hover {

color: red;

text-decoration: underline;

}

**2. :focus**

* The :focus pseudo-class is used to style elements when they receive focus, such as when a user clicks on an input field or navigates to it via the keyboard.
* **Example**:

input:focus {

border-color: blue;

background-color: lightyellow;

}

**3. :active**

* The :active pseudo-class applies to an element that is in the process of being clicked (i.e., when the mouse button is pressed down but not released).
* **Example**:

button:active {

background-color: green;

}

**4. :visited**

* The :visited pseudo-class is used to style links that have already been clicked by the user (i.e., the visited state of a hyperlink).
* **Example**:

a:visited {

color: purple;

}

**5. :first-child**

* The :first-child pseudo-class is used to select the first child of a specific type within a parent element.
* **Example**:

li:first-child {

font-weight: bold;

}

**6. :nth-child()**

* The :nth-child() pseudo-class targets elements based on their position in a parent element. You can use it with numbers, keywords, or formulas.
* **Example:-**

li:nth-child(odd) {

background-color: lightgray;

}

**4.2- Use of pseudo-classes to style elements based on their state:-**

CSS pseudo-classes allow you to style elements based on their state or interaction with the user.

1. :hover

2. :focus

3. :active

4. :visited

5. :first-child

6. :nth-child()

7. :checked

**5. CSS: ID and Class Selectors**

**5.1- Difference between id and class in CSS:-**

**1. ID in CSS:**

* **Unique:** An id is a unique identifier for an HTML element. Each id should only appear once in the document.
* Used to apply styling to one element.
* **Syntax :**-

# id{

Css declaration;

}

**2. Class in CSS:**

* **Reusable:** A class can be applied to multiple HTML elements. Unlike id, the same class can be used on multiple elements.
* Used to apply styling to multiple elements.
* Syntax:-

.class{  
 css declarations;

}

**5.2- Usage scenarios for id (unique) and class (reusable):-**

**1. id (Unique):**

An id is meant to uniquely identify a single element within a webpage. It should only be used once per document. The key use case for id is when you need to target a specific element that is unique on the page, and that element requires a distinct style or functionality

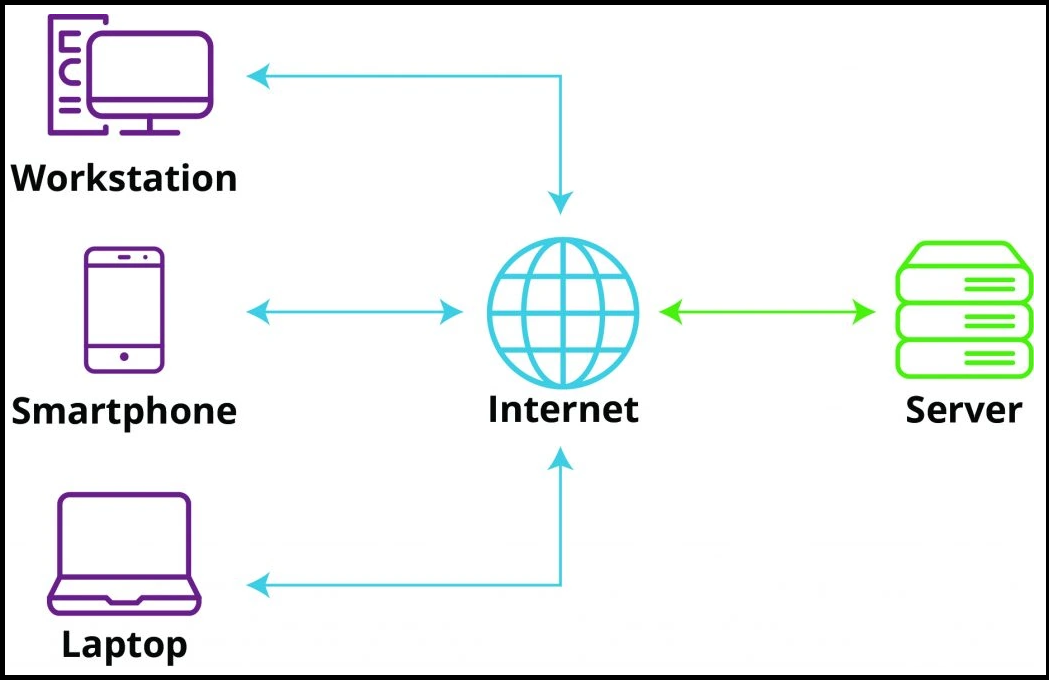
**2. class (Reusable)**

A class is used for grouping multiple elements that share the same styling or behavior. It can be applied to many elements across the page or even within different pages. The key use case for class is when you need to apply common styles or JavaScript functionality to multiple elements.

**6.** **Introduction to Client-Server Architecture**

**6.1- Overview of client-server architecture:-**

Client-server architecture is a design model for applications where tasks or workloads are divided between service providers (servers) and service requesters (clients). It is commonly used in network computing, where multiple clients interact with a centralized server to request services or resources.



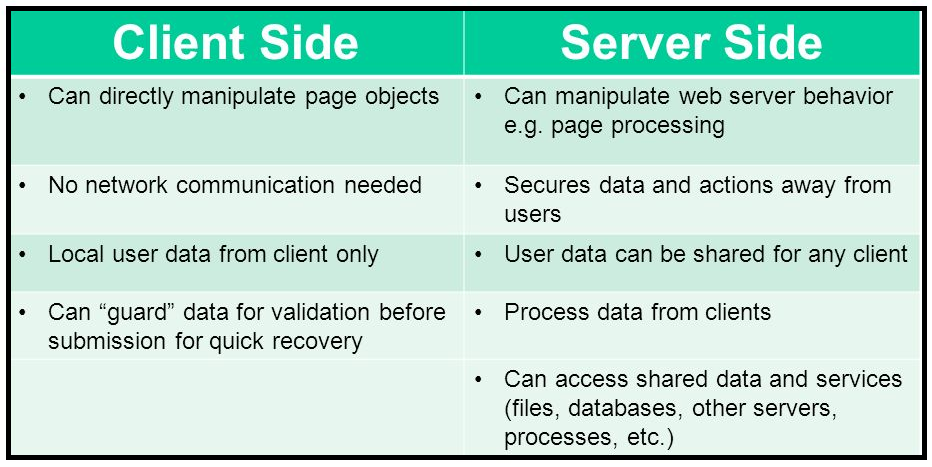
**6.2- Difference between client-side and server-side processing:-**

**1. Client-Side Processing**

Client-side processing occurs on the user's device (the client)usually a web browser or mobile app. In a typical web application, the client is the user's browser or app that interacts with the server.

**2. Server-Side Processing**

Server-side processing occurs on the server. The client sends requests to the server, which processes the request (e.g., fetching data from a database, performing complex computations) and sends the result back to the client.



**6.3- Roles of a client, server, and communication protocols:-**

**1. Roles of the Client**

The client is the entity that initiates requests for services, data, or resources. It is typically a device or application that interacts with the server over a network (e.g., a web browser, mobile app, or desktop application).

**2. Roles of the Server**

The server is the provider in the client-server architecture. It listens for requests from clients, processes them, and returns the appropriate responses. The server manages the resources and logic necessary to fulfill client requests.

**3. Roles of Communication Protocols**

Communication protocols define the rules, formats, and procedures for data exchange between the client and server. These protocols ensure that both parties understand how to structure, transmit, and interpret the data being exchanged.

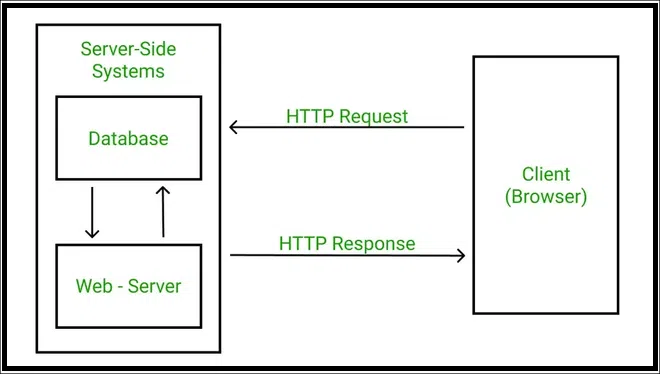
**7.HTTP Protocol Overview with Request and Response Headers**

**7.1- Introduction to the HTTP protocol and its role in web communication:-**

The HyperText Transfer Protocol (HTTP) is the foundation of data communication on the World Wide Web. HTTP is primarily used for retrieving resources, such as web pages, images, and files, from a server to display them to users in their web browsers.

**Role of HTTP in Web Communication:**

1. Enabling Web Browsing:
2. Enabling Web Browsing:
3. Supporting Web APIs:
4. Facilitating Communication Between Clients and Servers:
5. Secure Communication:

****

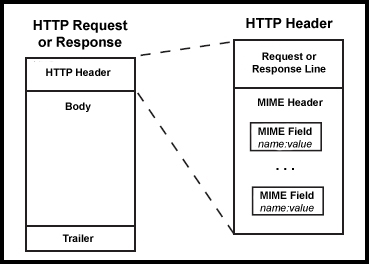
* 1. **Explanation of HTTP request and response headers:-**

**1. HTTP Request Headers**

An HTTP request is made when a client (like a browser) requests a resource (e.g., a webpage, an image, or data from an API) from a server. The headers in the request provide information about the client, the request itself, and any specifics the client wishes the server to know about the request.

**2. HTTP Response Headers**

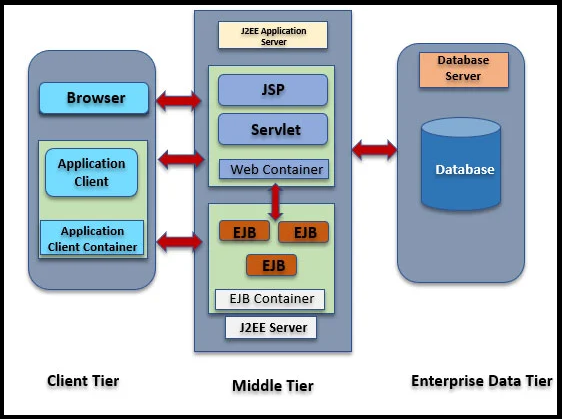
An HTTP response is sent by the server in reply to a client’s request. The response includes the requested data or an error message. Just like request headers, response headers carry metadata about the response, such as the type of content, status codes, caching instructions, and more.



1. **J2EE Architecture Overview**

**8.1- Introduction to J2EE and its multi-tier architecture:-**

A multi-tier architecture refers to the separation of different components of an application into distinct layers or tiers, which are responsible for specific tasks. This separation allows for easier development, maintenance, scalability, and improved security.



**8.2- Role of web containers, application servers, and database servers:-**

**1. Web Containers (Web Server or Servlet Container)**

A Web Container, also known as a Servlet Container, is a key component in J2EE (Jakarta EE) architecture responsible for managing the execution of web-based components, such as Servlets and JavaServer Pages (JSPs).

**2. Application Servers**

An Application Server is a more comprehensive server environment than a web container. It hosts both the business logic tier and integrates services for executing enterprise-level applications. In J2EE, the application server supports components such as Enterprise JavaBeans (EJBs), JSPs, Servlets, and other enterprise components.

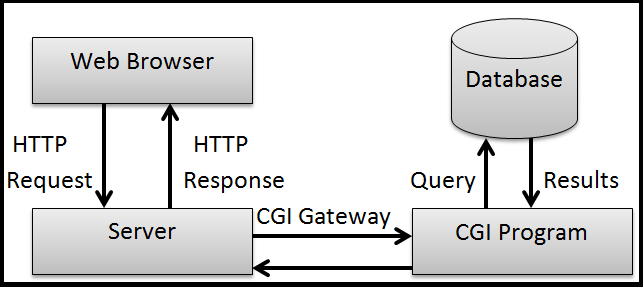
**3. Database Servers**

A Database Server is responsible for storing, retrieving, and managing the data used by the application. It forms part of the data tier of the multi-tier architecture. Database servers typically manage large amounts of structured data and provide robust features for data consistency, reliability, and performance.

**9.** **Web Component Development in Java (CGI Programming)**

**9.1- Introduction to CGI (Common Gateway Interface):-**

CGI (Common Gateway Interface) is a standard protocol that allows web servers to communicate with external programs or scripts in order to generate dynamic content on the web. CGI enables web servers to run scripts or executable programs (often referred to as CGI scripts) in response to user requests, typically from a web browser. This interaction allows the web server to send dynamic, customized content, such as user-specific data, processed forms, or results from databases.



**9.2- Process, advantages, and disadvantages of CGI programming.**

**Advantages:**

Language Flexibility: Works with various programming languages.

Simplicity: Easy to implement for dynamic content.

Portability: Compatible with different web servers.

**Disadvantages:**

Performance: Creates a new process for each request, which is inefficient.

Resource Intensive: Consumes CPU and memory with each request.

Security Risks: Vulnerable to attacks if not properly secured.

Scalability Issues: Hard to scale for high-traffic applications.

**10.** **Servlet Programming: Introduction, Advantages, and Disadvantages**

**10.1-** **History of servlet versions:-**

**Servlet 1.0 (December 1996):** Initial specification, part of the Java Servlet Development Kit (JSDK) 1.0.

**Servlet 2.0 (December 1997):** Continued development, part of JSDK 2.0.

**Servlet 2.1 (November 1998):** First official specification, included features like RequestDispatcher and ServletContext.

**Servlet 2.2 (August 1999):** Became part of J2EE introduced independent web application using .war files.

**Servlet 2.3 (2001):** Part of J2EE 1.3.

**Servlet 2.4 (2003):** Part of J2EE 1.4.

**Servlet 2.5 (2006):** Part of Java EE5.

**Servlet 3.0 (2009):** Introduced asynchronous processing and annotations.

**Servlet 4.0 (2017):** Included updates for Java EE 8 and removed deprecated methods.

**Servlet 5.0 (2018):** Renamed the javax package to Jakarta and introduced new features.

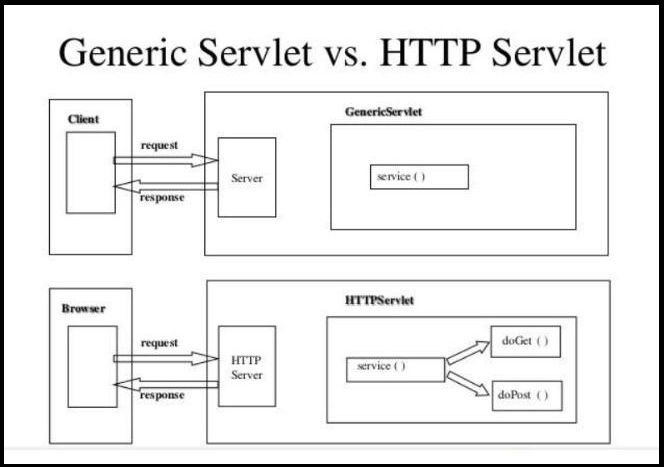
**10.2- Types of servlets: Generic and HTTP servlets:-**

**Generic Servlets:**

* A Generic Servlet is a general-purpose servlet that can handle any type of request (e.g., HTTP, FTP, etc.).
* It is part of the javax.servlet package and provides methods like service() to process requests.

**HTTP Servlets:**

* An HTTP Servlet is a specific type of servlet designed to handle HTTP requests.
* It extends the HttpServlet class and provides methods like doGet() and doPost() to handle GET and POST HTTP requests.



**11.** **Servlet Versions, Types of Servlets**

**11.1- Introduction to servlets and how they work**:-

A Servlet is a small Java program that runs on a web server and handles client requests, often generating dynamic web content like HTML.

**How Servlets Work:**

1. **User Request**: A user’s browser sends a request to the web server.
2. **Servlet Invocation**: The server sends the request to the Servlet.
3. **Processing**: The servlet processes the request (e.g., reading data, interacting with a database).
4. **Generate Response**: The servlet creates a response (usually HTML).
5. **Send Response**: The server sends the response back to the user’s browser to display.

**11.2- Advantages and disadvantages compared to other web technologies.:-**

Advantages of Servlets:

* Faster and more efficient than CGI.
* Scalable and handles many requests well.
* Built-in security with Java features.

Disadvantages of Servlets:

* More complex to code than simpler technologies.
* Requires knowledge of Java.
* Setting up a server can be more complicated.

**12.** **Difference between HTTP Servlet and Generic Servlet**

**12.1- Comparison between HttpServlet and GenericServlet:-**

|  |  |
| --- | --- |
| **HttpServlet** | **Generic Servlet** |
| 1. Http Servlet is Protocol dependent. | 1. Generic Servlet is Protocol independent. |
| 1. In Http Servlet we can define different logic for get() and post method. | 1. In Genric Servlet, we cannot define different logic for get() and post method |
| 1. Redirection is possible in http Serlet. | 1. Redirection is not possible in Generic Servelt. |
| 1. In http Servelet we can do Session management with Cookies. | 1. In Generic Servlet we cannot do session managent with Cookies. |
| 1. Session management with Http Session is also Possible in Http servlet | 1. Session management with Http Session is also not Possible in Generic Servlet. |
| 1. HttpServlet belongs to java.servlet.http package | 1. GenericServelet belongs to javax.servlet package |
| 1. HTTPServlet is a protocol dependent servlet | 1. Generic is a protocol independent servlet. |
| 1. HTTPServlet is an abstract class which extends GenericServlet and implements java.io.Serializable. | 1. Generic is an abstract class which extends Object and implements Servlet, ServletConfig and java.io.Serializable interfaces |
| 1. It is more commonly used in web server environment. | 1. It is not much useful in web server environment. |
| 1. HTTPServlet is extended by HTTP specific servlets. | 1. Generic is used for protocol independent servlets. |

**13.** **Difference between HTTP Servlet and Generic Servlet**

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

**1. init() Method**:

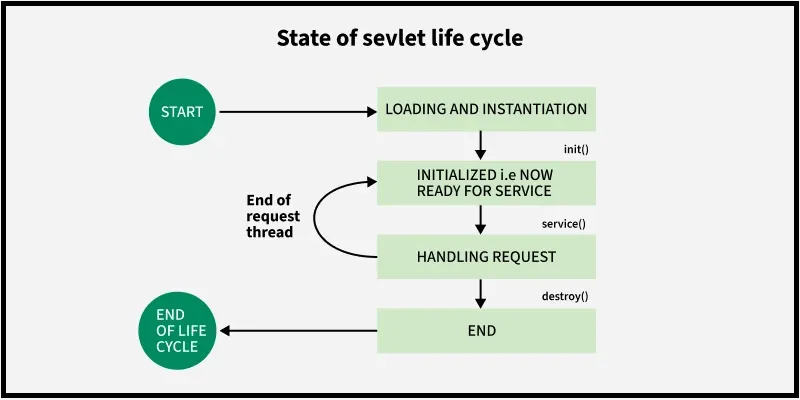
* Initializes the servlet when it’s first loaded.
* Once, when the servlet is loaded into memory.
* **Example**: Set up resources like database connections.

**2. service() Method**:

* Handles each client request (e.g., from a browser).
* Every time the servlet is requested.
* **Example**: Processes the request and sends a response.

**3. destroy() Method**:

* Cleans up when the servlet is no longer needed.
* When the servlet is being destroyed or the server is shutting down.
* **Example**: Close resources like database connections.



**14.** **Creating Servlets and Servlet Entry in web.xml**

* 1. **- How to create servlets and configure them using web.xml**

**1. Create a Servlet**

A servlet is a Java class that extends HttpServlet and overrides methods like doGet() or doPost() to handle HTTP requests.

**2. Configure the Servlet in web.xml**

In a web application, the web.xml file is used to define and configure servlets.

**15.** **Logical URL and ServletConfig Interface**

**15.1- Explanation of logical URLs and their use in servlets:-**

**What is a Logical URL?**

* A **logical URL** is a user-friendly address that maps to a servlet in your application. It is not tied to a specific file on the server but to a servlet's functionality.

**How Does it Work?**

* When you type a URL like http://localhost:8080/myApp/myServlet, the web server looks at the logical URL (/myServlet) and directs the request to the servlet that is mapped to this URL.

**15.2- Overview of ServletConfig and its methods.**

ServletConfig is an interface used in Java servlets to provide configuration information to the servlet. It allows a servlet to get initialization parameters and perform other configuration-related tasks when it is initialized by the servlet container.

**16.** **RequestDispatcher Interface: Forward and Include Methods**

**16.1- Explanation of RequestDispatcher and the forward() and include() methods:-**

**1. RequestDispatcher:**

* It is used to forward a request from one servlet to another servlet, JSP, or any other resource on the server.
* It allows a request to be passed along, meaning that the request can be processed by multiple servlets or JSPs.

**2. Methods of RequestDispatcher:**

**1. forward() method:**

* After a servlet processes a request, it can pass it to another resource for further processing or response generation.

**2. include() method:**

* Allows a servlet to include the output of another resource (like a servlet, JSP page, or HTML file) into its own response. Effectively, it performs a server- side include, where the output of the include resource is merged with the output of the current servlet.

**17.** **ServletContext Interface and Web Application Listener**

**17.1- Introduction to Servlet Context and its scope:-**

**Servlet Context:**

* The Servlet Context is an interface in Java that represents the entire web application. It provides a way for servlets to communicate with each other and share data across different servlets and JSPs.

**Scope of ServletContext:**

* The scope of ServletContext is application**-**wide. It means that any data stored in ServletContext is accessible to all servlets and JSPs in the same web application, and it remains available for the entire duration of the application’s life cycle (until the server is restarted or the application is stopped).

**17.2- How to use web application listeners for lifecycle events:-**

**1.ServletContextListener**:

* Responds to application (servlet context) lifecycle events (when the web application starts and stops)

**2.HttpSessionListener**:

* Responds to session lifecycle events (when a user session is created or destroyed).

**3.ServletRequestListener**:

* Responds to request lifecycle events (when a request is received and completed).

**18.** **Java Filters: Introduction and Filter Life Cycle**

**18.1- What are filters in Java and When are they needed?**

* A **filter** is an object that performs filtering tasks on either the **request to a resource** (such as a servlet or static content) or on the **response from a resource**.

**When are Filters Needed?**

Filters are typically used for:

* **Authentication** and **authorization** (e.g., checking if a user is logged in)
* **Logging** request and response information (e.g., for auditing)
* **Data compression** (e.g., compressing responses using GZIP)
* **Request modification** (e.g., preprocessing input data)
* **Response modification** (e.g., adding custom headers)
* **Image conversion** (e.g., scaling images)
* **Localization** (e.g., setting the correct language settings)

**18.2- Filter lifecycle and how to configure them in web.xml:-**

* **Basic Lifecycle of a Filter:**

1. **init()** — Called once when the filter is instantiated. Used for initialization.
2. **doFilter()** — Called every time a request/response pair is passed through the filter chain.
3. **destroy()** — Called once when the filter is taken out of service. Used for cleanup.

* **Filter Configuration in web.xml**

If you are not using annotations (@WebFilter), you configure filters manually in the web.xml deployment descriptor.

**It involves two parts:**

1. **Filter Declaration** — Define the filter class and a name.
2. **Filter Mapping** — Define **which URL patterns** or **servlets** the filter applies to.

**19.** **JSP Basics: JSTL, Custom Tags, Scriplets, and Implicit Objects.**

**19.1- Introduction to JSP and its key components: JSTL, custom tags, scriplets, and implicit objects.**

# Introduction to JSP (JavaServer Pages):-

* **JSP** stands for **JavaServer Pages**.
* It’s a **server-side technology** that helps you create **dynamic**, **data-driven web pages** easily.
* Instead of writing a full Java servlet, you can embed **Java code inside HTML** using JSP.
* JSPs are easier for designers and developers to work with compared to plain servlets because they mix HTML and Java code smoothly.
* **Key Components of JSP**

## 1. ****JSTL (JSP Standard Tag Library):-****

* JSTL provides **ready-made** tags for **common tasks** like:
  + Loops (forEach)
  + Conditionals (if)
  + Formatting (dates, numbers)
  + Database access (SQL tags)
* **Purpose**: Avoid writing Java code (like loops or if-else) inside JSP. Instead, use **tags**.

## 2. ****Custom Tags****

* You can create your **own custom tags** if JSTL is not enough.
* These custom tags are reusable and help organize complex JSP pages.
* You usually write them using **Tag Handlers** or with **Tag Files**.

**Example:** You can create a tag <my:hello/> that displays a welcome message.

## 3. ****Scriptlets****

* Scriptlets are Java code **directly embedded inside** JSP pages.
* **Syntax**: <% Java code %>

## 4. ****Implicit Objects****

JSP provides **predefined objects** you can use **directly** without creating them. These are called **implicit objects**.

**20.Session Management And Cookies.**

# 20.1- Overview of session management techniques:

**Session Management** in web applications is about **tracking user interactions** across multiple requests (since HTTP is **stateless** — it doesn’t remember users by default).We use several techniques to **identify and remember** users between different page requests.

# 1. ****Cookies****

* **Cookies** are small pieces of data stored **on the client’s browser**.
* Server sends a **Set-Cookie** header, and the browser automatically sends the cookie back on every request to that server.

**Example:**

* Server sets a cookie: Set-Cookie: sessionId=abc123
* Browser sends it with requests: Cookie: sessionId=abc123

# 2. ****Hidden Form Fields****

* You add a **hidden input field** inside a form that carries **session information** invisibly to the user.
* Every time the user submits a form, the hidden data is also submitted.

**Example**

<form action="nextPage.jsp">

<input type="hidden" name="sessionId" value="abc123">

<input type="submit" value="Continue">

</form>

# 3. ****URL Rewriting****

* Session data is **attached directly** to the **URL**.
* The server appends a unique ID or session token to the URL.

**Example**

String encodedURL = response.encodeURL("dashboard.jsp");

# 4. ****HTTP Sessions (Server-Side Sessions)****

* Most powerful and easiest technique.
* Server creates a **session object** and assigns a **unique session ID**.
* The session ID is typically stored in a **cookie** or appended to URLs.

**Example**

HttpSession session = request.getSession();

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

**20.2- How to track user sessions in web applications.**

* There are **four main ways** to track sessions:
  1. **Using Cookies**

**How it works:**

* Server creates a **session ID**.
* Sends it to the client inside a **Set-Cookie** header.
* Browser sends it back on every request.

## 2. ****Using URL Rewriting****

* If cookies are disabled, the server can **append the session ID to every URL** manually.

## 3. ****Using Hidden Form Fields****

* Add **hidden input fields** carrying the session ID inside forms.

## 4. ****Using HTTP Session (Server-Side Session Tracking)****

* The **server** creates a session object (HttpSession) for each user.
* The session data is stored **on the server**.
* A **session ID** is sent to the client (typically via a cookie) to match requests back to the correct session.