**Java Industry Assignment**

Module 3: Web Technologies in Java

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

* Introduction to HTML and its structure.
  + **HTML (HyperText Markup Language)** is the standard language used to create and design documents on the World Wide Web. It provides the basic structure for web pages and applications by using a series of elements and tags. These elements tell the web browser how to display text, images, links, and other content.
  + **Purpose of HTML**

HTML is not a programming language but a markup language. It allows developers to:

* Structure content into headings, paragraphs, lists, and sections.
* Insert multimedia elements such as images and videos.
* Create hyperlinks for navigation between pages.
* Build forms for user input.
* Basic Structure of an HTML Document
* A standard HTML document consists of several key parts, structured as follows:

html

CopyEdit

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Example Page</title>

</head>

<body>

<h1>This is a Heading</h1>

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

</body>

</html>

* Explanation of key tags:
* **<a> – Anchor Tag**
* Used to create **hyperlinks**, which allow users to navigate from one page to another.
* Syntax: CopyEdit<a href="https://www.example.com">Visit Example</a>
* href specifies the URL of the page the link goes to.
* **<form> – Form Tag**
* Defines an **HTML form** for user input.
* Can include elements like text fields, checkboxes, radio buttons, and submit buttons.
* Syntax: <form action="/submit" method="post">

<!-- Form elements go here -->

</form>

* **<table> – Table Tag**
* Used to display data in a **tabular format**.
* Usually includes <tr> (table row), <th> (table header), and <td> (table data).
* Example: <table>

<tr>

<th>Name</th>

<th>Age</th>

</tr>

<tr>

<td>Alice</td>

<td>20</td>

</tr>

</table>

* **<img> – Image Tag**
  + - Embeds **images** into a web page.
    - Syntax: <img src="image.jpg" alt="Description of image">
    - src specifies the image source, and alt provides alternative text.
* **List Tags: <ul>, <ol>, and <li>**
  + Used to create **lists**.
  + <ul>: Unordered (bulleted) list
  + <ol>: Ordered (numbered) list
  + <li>: List item
  + Example: html

<ul>

<li>Item One</li>

<li>Item Two</li>

</ul>

* **<p> – Paragraph Tag**
* Defines a **paragraph of text**.
* Adds space above and below the text block.
* <p>This is a paragraph of text.</p>
* **<br> – Line Break**
* Inserts a **line break** within text.
* It's an **empty tag** (no closing tag).
* Example:
* Line one.<br>Line two.
* **<label> – Label Tag**
  + Used to define **labels for form inputs**.
  + Improves accessibility and allows users to click on the label to focus the input.
  + Syntax:

<label for="username">Username:</label>

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

2.CSS: Inline CSS, Internal CSS, External CSS

* Overview of CSS and its importance in web design.
* **CSS (Cascading Style Sheets)** is used to style and format HTML content on web pages. It controls the appearance of elements such as colors, fonts, spacing, and layout.
* **Importance of CSS:**
* Enhances visual design of web pages
* Keeps HTML and styling separate for cleaner code
* Maintains consistent design across multiple pages
* Supports responsive design for mobile devices
* Makes websites easier to update and manage
* Types of CSS:
* **Inline CSS** – written inside HTML elements
* **Internal CSS** – written inside a <style> tag
* **External CSS** – written in a separate .css file

3.CSS: Margin and Padding

* Definition and difference between margin and padding.
* **Margin**
* Margin is the **space outside** an element's border. It creates space **between the element and other elements**.
* **Padding**
* Padding is the **space inside** an element’s border. It creates space **between the content and the border** of the element.
* How margins create space outside the element and padding creates space inside.
* **Margins** create space **outside** an element’s border. This means they push the element **away from other elements** on the page.
* **Padding** creates space **inside** the element’s border. It pushes the content **away from the border**, adding space **around the content** inside the element.

4.CSS: Pseudo-Class

* Introduction to CSS pseudo-classes like :hover, :focus, :active, etc.
* **CSS pseudo-classes** are used to define a special state of an element. They allow you to apply styles based on user interaction or element position without adding extra classes or JavaScript.
* **Common Pseudo-classes:**
* **:hover** – Applies styles when the user **hovers** the mouse over an element.

css

CopyEdit

a:hover {

color: red;

}

* **:focus** – Applies styles when an element, like an input, is **focused** (clicked or selected).

css

CopyEdit

input:focus {

border: 2px solid blue;

}

* **:active** – Applies styles when an element is **being clicked**.

css

CopyEdit

button:active {

background-color: green;

}

* Targets the **first child** of a parent element.

css

CopyEdit

p:first-child {

font-weight: bold;

}

* **:last-child** – Targets the **last child** of a parent element.
* Use of pseudo-classes to style elements based on their state.
  + **CSS pseudo-classes** are used to apply styles to elements based on their **state or position** without adding extra classes or JavaScript.
  + **Why It's Useful:**
* Improves **user interaction**
* Makes web pages more **dynamic and responsive**
* Enhances **accessibility and usability**

5. CSS: ID and Class Selector

* Difference between id and class in CSS

|  |  |  |
| --- | --- | --- |
| **Features** | **Id** | **class** |
| **Symbol** | # (hash) | . (dot) |
| **Uniqeness** | Must be **unique** (used once per page) | Can be used **multiple times** |
| **Usage** | Targets **one specific** element | Targets **a group** of elements |
| **Example** | #header { color: blue; } | .box { color: red; } |

* Usage scenarios for id (unique) and class (reusable).
* **When to Use id (Unique):**
* Styling a **single, specific** element on a page.
* Targeting a unique section like a header, footer, or main container.
* Linking to a specific part of the page using anchor links.
* **When to Use class (Reusable):**
  + - Styling **multiple elements** with the same style.
    - Applying the same layout or color to buttons, cards, or sections.
    - Reusing styles across different parts of the webpage.

6. Introduction to Client-Server Architecture

* Overview of client-server architecture.
* Client-Server Architecture is a network model where **clients** and **servers** communicate with each other.
* A **client** is a computer or application (like a browser) that sends a request.
* A **server** is a powerful system that processes the request and sends back a response
* How it works:

1. Client sends a request to the server (e.g., open a web page).
2. Server processes the request.
3. Server sends the response (like HTML content) back to the client.

* Example:
  + When you open a website, your **browser (client)** sends a request, and the **web server** sends back the page.
* Benefits:
* Centralized control
* Easy to update and manage

|  |  |  |
| --- | --- | --- |
| **Feature** | **Client-side Processing** | **Server-side Processing** |
| **Where it runs** | On the user's browser | On the web server |
| **Languages used** | HTML, CSS, JavaScript | PHP, Python, Node.js, Java, etc. |
| **Speed** | Faster (no need to contact server) | Slower (requires server response) |
| **Security** | Less secure (code is visible to users) | More secure (code runs on server) |
| **Use cases** | Form validation, animations, UI effects | Database access, login systems, file handling |

* Better security and performan
* Difference between client-side and server-side processing.
* Roles of a client, server, and communication protocols.

1. Client:
   * + The **client** is a user's device or application (like a browser).
     + It **sends requests** to the server for data or services.
     + It **displays** the received data to the user.
   1. Server:
      * The **server** is a powerful computer or system.
      * It **receives and processes** client requests.
      * It **sends responses** back to the client (like web pages, files, or data).
   2. Communication Protocols:
      * Protocols are rules that define how data is exchanged.
      * Common protocols:
        + **HTTP/HTTPS** – For web communication
        + **FTP** – For file transfers
        + **TCP/IP** – For reliable data transfer over the internet

7. HTTP Protocol Overview with Request and Response Headers

* Introduction to the HTTP protocol and its role in web communication
* **HTTP (HyperText Transfer Protocol)** is the foundation of data communication on the web. It defines how messages are **formatted, sent, and received** between a client (like a web browser) and a server.
* **Role of HTTP in Web Communication:**
* The **client** sends an **HTTP request** to the server (e.g., asking for a webpage).
* The **server** processes the request and sends back an **HTTP response** (e.g., HTML, CSS, images).
* This allows users to **view and interact with websites**.

1. **Examples:**

* Your browser sends an HTTP request.
* The server responds with the website’s content.
* Explanation of HTTP request and response headers
* **HTTP headers** are key-value pairs sent with requests and responses to provide **extra information** about the communication between the client and server.

1. Request Headers:

Sent from **server to client**. They describe the response.

**Common response headers:**

* Content-Type: Type of data (e.g., text/html, application/json)
* Set-Cookie: Sends cookies to be stored in the browser
* Cache-Control: Controls how and when to cache the content
* Server: Information about the server software

1. Response Headers:

Sent from **server to client**. They describe the response.

**Common response headers:**

* Content-Type: Type of data (e.g., text/html, application/json)
* Set-Cookie: Sends cookies to be stored in the browser
* Cache-Control: Controls how and when to cache the content
* Server: Information about the server software

8. J2EE Architecture Overview

* Introduction to J2EE and its multi-tier architecture
* What is J2EE?
* **J2EE (Java 2 Platform, Enterprise Edition)** is a platform developed by **Sun Microsystems** (now Oracle) for building **enterprise-level web applications** using the Java programming language.
* It provides a set of APIs and tools to create **scalable, secure, and multi-tiered applications**.
* Multi-Tier Architecture in J2EE:
* J2EE applications are typically divided into **three main tiers**:

**1. Client Tier:**

* + This is the front end.
  + It includes web browsers or mobile apps used by users to interact with the system.

**2. Web Tier:**

* Handles **user requests** and **responds** with the right web pages.
* Uses **Servlets, JSP (JavaServer Pages)** to process and display content.

**3. Business Tier:**

* Contains the **business logic** of the application.
* Uses **Enterprise JavaBeans (EJB)** to perform operations like calculations, processing orders, etc.

**4. Data Tier (Enterprise Information System Tier):**

* Responsible for **storing and retrieving data**.
* Communicates with **databases** using JDBC (Java Database Connectivity).
* Role of web containers, application servers, and database servers.

1. **Web Container (Servlet Container):**

* Runs **web components** like Servlets and JSP (JavaServer Pages).
* Manages the **life cycle** of web components.
* Handles **HTTP requests and responses**.
* Example: **Apache Tomcat**

1. **Application Server:**

* Provides a full environment to run **enterprise applications**.
* Supports **business logic**, transactions, security, and web services.
* Can include a web container and EJB container.
* Example: **GlassFish**, **WildFly**, **WebLogic**

1. **Database Server:**
   * + Stores and manages **data** used by the application.
     + Responds to queries from the application server using **SQL**.
     + Example: **MySQL**, **Oracle Database**, **PostgreSQL**

9. Web Component Development in Java (CGI Programming)

* Introduction to CGI (Common Gateway Interface)
* **CGI (Common Gateway Interface)** is a standard protocol used to enable **web servers** to interact with **external programs** (called CGI scripts) to generate **dynamic content** on web pages.
* **How CGI Works?**

1. A user sends a request from a browser (e.g., submitting a form).
2. The web server passes the request to a CGI script (usually written in Python, Perl, or C).
3. The script processes the input and sends back the output (like HTML) to the server.
4. The server sends the result to the browser.

* **Features:**
  + - Used to create **dynamic web pages**
    - Language-independent (supports many languages)
    - Was one of the **earliest ways** to handle server-side processing.
  + **Example Use:**
    - Feedback forms
    - Login systems
    - Search queries
* Process, advantages, and disadvantages of CGI programming.
* **Process of CGI Programming:**
  1. User submits a form or request from a web browser.
  2. Web server receives the request and sends it to a **CGI script**.
  3. The CGI script processes the input (e.g., saves data, checks login).
  4. Script sends output (usually HTML) back to the server.
  5. Web server displays the result to the user in the browser.
* **Advantages of CGI:**
* Simple and easy to set up
* Language independent (works with Python, Perl, C, etc.)
* Good for small-scale, dynamic web content
* Supported by most web servers
* **Disadvantages of CGI:**
  + Slower performance (creates a new process for each request)
  + Not suitable for high-traffic websites
  + Limited scalability
  + Older technology – newer methods (like PHP, JSP, Node.js) are more efficient

10. Servlet Programming: Introduction, Advantages, and Disadvantage

* Introduction to servlets and how they work.
* **What is a Servlet?**
* A Servlet is a Java program that runs on a web server and is used to create dynamic web content. Servlets handle client requests (like form submissions) and send responses (like HTML pages).
* **How Servlets Work:**

1. A client (browser) sends a request to the server.
2. The **web container** (like Tomcat) receives the request.
3. The request is passed to the **Servlet**.
4. The Servlet processes the request (e.g., read data, access database).
5. The Servlet sends back a **response** (like a web page) to the client.

* **Features:**
  + Written in **Java**
  + More efficient than CGI (no new process for each request)
  + Platform-independent
  + Reusable and easy to maintain
* Advantages and disadvantages compared to other web technologies.
* **Advantages of Servlet:**
* **Efficient:** Unlike CGI, Servlets don’t create a new process for each request — they use threads, which saves memory and time.
* **Platform-independent:** Written in **Java**, so they run on any system with a Java-enabled server.
* **Secure:** Supports built-in security features like HTTPS and session management.
* **Reusable and maintainable:** Easy to update and reuse code.
* **Integrated with Java APIs:** Can easily connect to databases, work with XML, and more.
* **Disadvantages of Servlet:**
* **More complex** than scripting languages like PHP or Python.
* Requires a **Java-enabled server** (like Tomcat or Glassfish).
* Not suitable for simple or small websites — overkill for basic tasks.
* Steeper **learning curve** for beginners compared to HTML/PHP.

11. Servlet Versions, Types of Servlets

* History of servlet versions
* **Servlet 1.0 (1997):** Basic request & response handling
* **Servlet 2.2 (1999):** Introduced web applications and WAR files
* **Servlet 2.3:** Added filters and event listeners
* **Servlet 2.5:** Supported annotations and simplified config
* **Servlet 3.0:** Added asynchronous processing and annotations
* **Servlet 3.1:** Non-blocking I/O for better performance
* **Servlet 4.0:** Supports HTTP/2 (faster communication)
* Types of servlets: Generic and HTTP servlets.

1. **Generic Servlet:**

* A general-purpose servlet.
* It belongs to javax.servlet.GenericServlet class.
* Not specific to any protocol.
* Can be used for custom protocols (rarely used in web apps).

**Example use:** Basic message handling, logging.

1. **HTTP Servlet:**

* A servlet designed for handling **HTTP requests**.
* It extends javax.servlet.http.HttpServlet class.
* Used in most **web applications**.
* Supports methods like doGet(), doPost().

**Example use:** Handling form submissions, login systems.

12. Difference between HTTP Servlet and Generic Servlet

* Detailed comparison between HTTP Servlet and Generic Servlet.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Generic Servlet** | **HTTP Servlet** |
| **Class Location** | javax.servlet.GenericServlet | javax.servlet.http.HttpServlet |
| **Protocol Support** | **Protocol-independent** (can be used for any protocol) | **Designed for HTTP protocol** only |
| **Common Use** | Rarely used in web apps | Widely used in web-based Java applications |
| **Main Method** | Overrides the service() method | Overrides doGet(), doPost(), doPut(), doDelete() etc. |
| **Flexiblity** | More flexible, can handle different types of requests | Limited to HTTP request handling |
| **Ease of Use for Web Apps** | Not very convenient for HTTP requests | Built specifically for **web (HTTP)** usage |
| **Use Case example** | Low-level custom server apps or protocols | Handling form submissions, login systems, APIs, etc. |
| **Inheritance Hierarchy** | Implements Servlet interface directly | Inherits from GenericServlet internally |

13. Servlet Life Cycle

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

1. **init() Method**

* Called **once** when the servlet is first loaded.
* Used for **initial setup**, like opening a database connection.
* Called **only once** during the servlet's lifetime.

1. **service () Method**

* Called **every time** the servlet receives a request.
* Handles the logic and sends the response.
* In HttpServlet, this method calls doGet(), doPost(), etc.javaCopyEdit

1. **destroy() Method**

* Called **once** before the servlet is removed from memory.
* Used for **cleanup** tasks, like closing database connections.

14. Creating Servlets and Servlet Entry in web.xml

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

**Step 1: Create a Servlet Class**

public class HelloServlet extends HttpServlet {

public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException {

res.setContentType("text/html");

res.getWriter().println("<h2>Hello from Servlet!</h2>");

}

}

**Step 2**: **Configure Servlet in web.xml**

<servlet>

<servlet-name>HelloServlet</servlet-name>

<servlet-class>HelloServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloServlet</servlet-name>

<url-pattern>/hello</url-pattern>

</servlet-mapping>

**Step 3: Run the Servlet**

<http://localhost:8080/YourAppName/hello>

15. Logical URL and ServletConfig Interface

* Explanation of logical URLs and their use in servlets.
* A **logical URL** is a **custom, user-defined path** used to access a servlet, instead of using the actual class name. It is defined in web.xml or by annotations.

**Example:**

**In web.xml:**

<servlet-mapping>

<servlet-name>MyServlet</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

Now you can access the servlet using:

<http://localhost:8080/YourAppName/welcome>

**Uses:**

* Makes URLs easy to read and remember
* Hides internal code structure from the user
* Helps in better organization and security
* Allows easy navigation in web apps
* Overview of Servlet Config andits methods.
* **What is ServletConfig?**
* ServletConfig is an **interface** used to pass **initialization parameters** to a servlet. These parameters are defined in the web.xml file.
* It allows each servlet to access its own config info.
* **Common Methods of ServletConfig:**

|  |  |
| --- | --- |
| **Method** | **Description** |
| getInitParameter(String name) | Returns the value of a specific init parameter |
| getInitParameterNames() | Returns all init parameter names as an enum |
| getServletName() | Returns the name of the servlet |
| getServletContext() | Returns the ServletContext object |

16. RequestDispatcher Interface: Forward and Include Methods

* Explanation of RequestDispatcher and the forward() and include() methods.
* **What is RequestDispatcher?**

RequestDispatcher is an interface in Java used to **forward a request** from one servlet to another resource (like a servlet, JSP, or HTML file) or to **include** another resource in the response.

* **Methods:**

1. forward (Request, Response)

* **Transfers control** to another resource.
* The original servlet stops processing.
* **URL doesn't change** in the browser.

1. Include (request, response)

* **Includes** the content of another resource in the current response.
* The original servlet continues after including.

17. Servlet Context Interface and Web Application Listener

* Introduction to ServletContext and its scope.
* **What is ServletContext?**
* ServletContext is an object provided by the **web container** that allows servlets to **communicate with the server** and **share information** across the whole web application.
* It is **created once** when the application starts and is shared by **all servlets**.
* **Scope of ServletContext:**
* **Application-wide scope** (available to all servlets and JSPs)
* Used to **store data** that needs to be shared across the app
* Lives from app start to app shutdown
* How to use web application listeners for lifecycle events.
* **What are Listeners?**

Listeners are special classes in Java that monitor events in a web application — like when the app starts or stops, or when sessions are created/destroyed.

* **Types of Lifecycle Listeners:**

1. **ServletContextListener** – Monitors **application startup and shutdown**
2. **HttpSessionListener** – Monitors **session creation and destruction**
3. **ServletRequestListener** – Monitors **request events**

18. Java Filters: Introduction and Filter Life Cycle

* What are filters in Java and when are they needed?
* What Are Filters?
* **Filters** in Java are special components that **intercept requests and responses** in a web application before they reach servlets or JSPs.
* They belong to the javax.servlet package and implement the Filter interface.
* When Are Filters Needed?

Filters are used when you want to perform tasks **before or after** servlet processing. Common uses include:

* Authentication and Authorization
* Input Validation or Data Filtering
* Logging and Auditing
* Compression (like GZIP)
* Blocking certain requests
* Filter lifecycle and how to configure them in web.xml.
* **Filter Lifecycle:**

1. **init()**

* Called **once** when the filter is first created.
* Used for initialization (like reading config).

1. **doFilter()**

* Called **every time** a request matches the filter mapping.
* Main logic is written here (pre-processing and post-processing).

1. **destroy()**

* Called **once** when the filter is removed.
* Used for cleanup tasks.
* **Example Filter Code:**

public class MyFilter implements Filter {

public void init(FilterConfig config) { /\* setup code \*/ }

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)

throws IOException, ServletException {

// pre-processing

chain.doFilter(req, res); // pass to next filter/servlet

// post-processing

}

public void destroy() { /\* cleanup code \*/ }

}

* **How to Configure Filter in web.xml:**

<filter>

<filter-name>MyFilter</filter-name>

<filter-class>com.example.MyFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>MyFilter</filter-name>

<url-pattern>/\*</url-pattern> <!-- Applies to all URLs -->

</filter-mapping>

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

* **What is JSP?**

JSP (JavaServer Pages) is a **server-side technology** that allows writing **dynamic web pages** using Java and HTML together. It is easier than servlets for designing UI-based web applications.

* **Key Components of JSP:**

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

* A set of ready-to-use **tags for common tasks** like loops, conditions, formatting, etc.
* Makes JSP pages cleaner and avoids Java code in HTML.
* <c:forEach var="item" items="${list}">...</c:forEach>

1. **Custom Tags:**

* User-defined tags created using tag handler classes.
* Help separate business logic from presentation.

1. **Scriplets (<% %>):**

* Allow writing raw **Java code inside JSP**.
* Not recommended now (replaced by JSTL and EL).
* <% int x = 5; %>

1. **Implicit Objects:**

* Predefined objects in JSP like:
* request, response, session, application, out, config, pageContext, etc.
* Used to access request data, manage session, write output, etc.
* <%= request.getParameter("username") %>

20. Session Management and Cookies

* Overview of session management techniques: cookies, hidden form fields, URL rewriting, and sessions.
* Session management is used to **track user data** across multiple requests in a web application. There are several techniques to manage sessions:
  1. **Cookies:**
* Small text files stored on the **client's browser**.
* Automatically sent with every request to the server.
* Used to store session ID or user preferences.
  1. **Hidden Form Fields:**
* Data is stored in **hidden HTML input fields**.
* Sent with the form submission to the server.
* Visible in page source; not secure.
  1. **URL Rewriting:**
* Session data is added to the **URL as a parameter**.
* Useful when cookies are disabled.
  1. **HttpSession:**
* Server-side technique using HttpSession object.
* Stores data on the **server**, unique to each user.
* More secure and commonly used.
* How to track user sessions in web applications.
* User sessions are tracked to maintain user-specific data (like login info, cart items) across multiple requests. There are several ways to do this:
  1. Using HttpSession (Most Common)
* java provides HttpSession to store data on the **server-side**.
* unique session ID is created for each user.
  1. Using Cookies
* A small file stored in the **client’s browser**.
* Session ID or user info can be saved and sent with every request.
  1. Using URL Rewriting
* Adds session ID or data in the **URL** if cookies are disabled.
  1. Using Hidden Form Fields
* Data is stored in a hidden input field and submitted with forms.