Web Technologies in Java

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

1.1.Introduction to HTML and its structure.

Ans : -

HTML (HyperText Markup Language) is the standard language used to create and design web pages. It tells the browser what to display (text, images, links, forms, etc.). HTML is not a programming language; it's a markup language that uses tags to structure content. Every web page you see online is built with HTML as its foundation.

```
<!DOCTYPE html>
<html>
 <head>
    <title> Page Title </title>
 </head>
<body>
   <h1>Heading 1 </h1>
   <p>Paragraph text </p>
</body>
</html>
```

1.2 Explanation of key tags :

Ans

1 <!DOCTYPE html> – Defines HTML5 document type.

2 <html> – Root of an HTML document.

3 <head> – Contains meta info, title, CSS, etc.

4 <title> – Sets page title (shown in browser tab).

5 <body> – Holds all visible content.

6 <h1>–<h6> – Headings (from largest to smallest).

7 <p> – Paragraph text.

8 <a> – Hyperlink.

9 <img> – Displays image.

10 <ul> / / – Lists (unordered/ordered/items).

11 <table>, <tr>, <th>, <td> – Table structure.

12 <form> – Creates form for input.

13 <input> – Input field (text, button, etc.).

14 – Block-level container.

15 – Inline text container.

16
 – Line break.

17 <hr> – Horizontal line.

18 <header>, <footer>, <section>, <nav> – Page layout elements.

19 <link> – Connects external CSS file.

20 <script> – Adds JavaScript.

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

2.1.Overview of CSS and its importance in web design.

CSS (Cascading Style Sheets) is used to **style and design web pages**.
It controls how HTML elements look — colors, fonts, layout, spacing, etc.

Importance:

- Makes websites **attractive and user-friendly**.
- Separates **content (HTML)** from **design (CSS)**.
- Enables **faster page loading** and **easier maintenance**.
- Allows **responsive design** (works on all screen sizes).
- Promotes **consistency** across multiple pages.

2. Types of CSS

Type	Description	Example
Inline CSS	CSS is written **inside the HTML tag** using the `style` attribute.	`<h1 style="color:red;">Hello</h1>`
Internal CSS	CSS is written **inside `<style>` tag** within the `<head>` section of the same HTML file.	`<style>h1{color:blue;}</style>`
External CSS	CSS is written in a **separate .css file** and linked using `<link>` tag.	`<link rel="stylesheet" href="style.css">`

3. CSS Margin and Padding

3.1 .Definition and difference between margin and padding.

Definition and Difference:

- **Margin:** Space **outside** an element — creates distance between the element and others.
- **Padding:** Space **inside** an element — creates distance between the element's content and its border.

Example:

```
div {
  margin: 20px;   /* space outside */
  padding: 10px;  /* space inside */
}
```

Summary:
Margin = Outside space
Padding = Inside space

4. CSS: Pseudo-Class.

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

Definition:
A **pseudo-class** in CSS is used to define a **special state of an element** — for example, when a user hovers over a button or focuses on an input field.

Common Pseudo-Classes:

- :hover → When the mouse is over an element.
- :focus → When an element (like input) is focused.
- :active → When an element is being clicked.
- :visited → For visited links.
- :first-child → Targets the first child of a parent element.

```
button:hover {
  color: blue;
  color: white;
}
```

Pseudo-classes style elements based on user interaction or position in the document.

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

Explanation:
Pseudo-classes are used to **change the style of elements depending on their state or interaction** — like when a user hovers, clicks, or focuses on them.

```
a:hover {
  color: red;       /* when mouse is over the link */
}
button:active {
  background: green; /* when button is clicked */
}
input:focus {
  border-color: blue; /* when input box is selected */
}
```

Pseudo-classes help make web pages interactive and responsive to user actions.

5.CSS: ID and Class Selectors.

5.1 Difference between id and class in CSS.

Definition:

- **ID Selector (#)** – Used to style a **single, unique element**.
- **Class Selector (.)** – Used to style **multiple elements** with the same design.

Syntax Example:

```
#header {
  color: blue;   /* ID selector */
}
.title {
  color: red;    /* Class selector */
}
```

Difference between ID and Class:

Basis	**ID**	**Class**
Symbol	#	.
Uniqueness	Used for **one element only**	Used for **multiple elements**
Priority	**Higher** in CSS	**Lower** than ID
Example	#header {}	.title {}

5. 2.Usage scenarios for id (unique) and class (reusable).

ID (#) – Unique Usage:

Used for **styling or targeting a single, unique element** on a page. Example: Header, Footer, or a specific section.

Welcome!

Class (.) – Reusable Usage:
Used for **styling multiple elements** with the same design or behavior. Example: Buttons, cards, or text styles.

<button class="btn">Submit</button>

<button class="btn">Cancel</button>

6.Introduction to Client-Server Architecture

6. 1.Overview of client-server architecture.

Client-Server Architecture is a **network model** where two main components — **client** and **server** — communicate with each other.

- The **client** sends a **request** for data or service.
- The **server processes the request** and sends back a **response**.

Example:
When you open a website —

- Your **browser (client)** sends a request to the **web server**.
- The **server** responds with the webpage data.

Client = requester of service
Server = provider of service

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

Basis	**Client-Side Processing**	**Server-Side Processing**
Where it runs	On the **user's browser**	On the **web server**
Languages used	HTML, CSS, JavaScript	Java, PHP, Python, Node.js, etc.

Speed	Faster (no server communication needed)	Slower (needs server response)
Data Access	Cannot access database directly	Can access and modify database
Example	Form validation using JavaScript	Login verification using database

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

Ans:

1 Client: Requests services or resources from the server. Examples: Web browser, mobile app.

2.Server: Provides services, processes client requests, and sends back responses. Examples: Web server, database server.

3.Communication Protocols: Rules that define how clients and servers exchange data. Examples: HTTP/HTTPS for web, FTP for file transfer, SMTP for email.

7.HTTP Protocol Overview with Request and Response Headers

7.1 HTTP Protocol Overview with Request and Response Headers

Ans : -

HTTP (HyperText Transfer Protocol) is the **standard protocol** used for communication between a **web client (browser)** and a **web server**.

Role in Web Communication:

1. **Client sends a request** – e.g., asks for a web page.
2. **Server processes the request** – retrieves the required page or data.
3. **Server sends a response** – returns the webpage, data, or an error message to the client.

2. Explanation of HTTP Request and Response Headers

HTTP Headers are **key-value pairs** exchanged between the client and server that provide **extra information** about the request or response.

- **Request Headers:**
  Sent **by the client to the server**, containing details about the request.
  Examples:
  - Host – Domain name of the server ○ User-Agent – Browser or client details ○ Accept – Type of content client can handle
- **Response Headers:**
  Sent **by the server to the client**, containing details about the response.
  Examples:
  - Content-Type – Type of data sent (e.g., text/html, application/json) ○ Content-Length – Size of the response data ○ Server – Information about the server software

8. J2EE Architecture Overview.

8. 1.Introduction to J2EE and its multi-tier architecture.

J2EE (Java 2 Platform, Enterprise Edition) *supports* ***multi-tier architecture****, which divides an application into separate layers to improve* ***scalability, maintainability, and flexibility****.*

Multi-Tier Architecture Layers:

1. ***Client Tier (Presentation Layer)*** ○ *Provides the* ***user interface*** *and interacts with users.* ○ ***Examples:*** *Web browsers, desktop applications.*
2. ***Web Tier (Presentation Logic)*** ○ *Handles* ***client requests and responses****.*
   - ○ ***Examples:*** *Servlets, JSPs.*

3. ***Business Tier (Business Logic Layer)*** ○ *Contains* ***business rules and logic*** *of the application.*
    - ○ ***Examples:*** *EJBs (Enterprise Java Beans), Java classes.*
4. ***Enterprise Information System (EIS) Tier (Data Layer)*** ○ *Manages* ***data storage and retrieval****.*
    - ○ ***Examples:*** *Databases, legacy systems.*

8.2 Role of Web Containers, Application Servers, and Database Servers

1. ***Web Container*** ○ *Manages* ***web components*** *like Servlets and JSPs.* ○ *Handles* ***HTTP requests and responses****.*
    - ○ ***Examples:*** *Apache Tomcat, Jetty.*
2. ***Application Server*** ○ *Manages* ***business logic components*** *like EJBs.*
    - ○ *Provides enterprise services such as* ***transactions, security, and messaging****.*
    - ○ ***Examples:*** *JBoss, WebLogic, GlassFish.*
3. ***Database Server*** ○ *Stores and manages* ***application data****.*
    - ○ *Handles* ***queries, updates, and ensures data integrity****.*
    - ○ ***Examples:*** *MySQL, Oracle, SQL Server.*

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 a* ***web server to interact with external programs or scripts*** *to generate* ***dynamic web content****.*

Key Points:

- *Enables web pages to **respond to user input**.*
- *CGI programs can be written in **Perl, Python, C, or Java**.*

Working Process:

1. ***Client sends a request** to the web server.*
2. ***Server executes the CGI program**.*
3. ***CGI program processes data** and sends a **dynamic response** back to the client.*

9.2 Process, Advantages, and Disadvantages of CGI Programming Process

of CGI Programming:

1. ***Client Request:** User submits data via a form or URL.*
2. ***Server Executes CGI Program:** The web server runs the external script or program.*
3. ***Processing:** The CGI program processes the input and generates output.*
4. ***Server Response:** The generated HTML or data is sent back to the client's browser.*

Advantages of CGI:

- *Simple and **easy to implement**.*
- ***Widely supported** by web servers.*
- *Can be written in **multiple languages** (Perl, Python, C, Java, etc.).*
- *Enables **creation of dynamic and interactive web pages**.*

Disadvantages of CGI:

- ***Slower performance** – each request starts a new process.*
- ***High server load** for many simultaneous users.*

- ***Difficult to scale*** *for large web applications.*

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

10.1 Introduction to Servlets and How They Work

Servlets *are* ***Java programs*** *that run on a* ***web server*** *and are used to create* ***dynamic web content****.*
They act as a ***middle layer*** *between a* ***client request*** *(usually from a web browser) and the* ***server-side resources*** *(like databases or files).*

How Servlets Work:

1. ***Client sends a request*** *to the web server.*
2. ***Web server forwards the request*** *to the* ***servlet container*** *(like Tomcat).*
3. ***Servlet processes the request****, performs logic or database operations.*
4. ***Servlet generates a response*** *(usually HTML) and sends it back to the client.*

Example:
When a user submits a login form, a servlet can check the credentials and return a success or error page.

10.2 Advantages and Disadvantages of Servlets Compared to Other Web Technologies

Advantages:

1. ***Platform Independent*** *– Written in Java, so they run on any server with a JVM.*
2. ***Efficient and Fast*** *– Runs within the server process (no new process for each request like CGI).*
3. ***Reusable and Maintainable*** *– Uses Java classes and OOP concepts.*
4. ***Secure*** *– Supports HTTPS, session tracking, and Java's security features.*

5. ***Integration*** *– Can easily connect with databases and other Java technologies (JSP, JDBC, etc.).*

Disadvantages:

1. ***Complex for Large HTML Outputs*** *– Writing HTML inside Java code is difficult.*
2. ***Requires Java Knowledge*** *– Not suitable for developers unfamiliar with Java.*
3. ***More Setup Needed*** *– Needs a servlet container (like Tomcat) to run.*

11.Servlet Versions, Types of Servlets.

11.1 History of Servlet Versions

Servlet *technology evolved with different versions to add new features and improvements for web development in Java.*

- **1.0 (1997):** Basic API for request/response.
- **2.0 (1998):** Added ServletContext, ServletConfig.
- **2.1 (1999):** Introduced RequestDispatcher, web.xml.
- **2.2 (2000):** WAR files and web app concept.
- **2.3 (2001):** Added Filters and Listeners.
- **2.4 (2003):** XML Schema–based web.xml.
- **2.5 (2005):** Annotation support (@WebServlet).
- **3.0 (2009):** Full annotations, async support.
- **3.1 (2013):** Non-blocking I/O.
- **4.0 (2017):** HTTP/2 support.
- **5.0 (2020):** Moved to jakarta.servlet package.
- **6.0 (2022):** Jakarta EE 10 features.
- **6.1 (2023):** Latest, Jakarta EE 11, improved async and HTTP handling.

11.2 Types of Servlets: Generic and HTTP Servlets

1. ***Generic Servlet:***

   *An **abstract class** that implements the Servlet interface.*

   *Can handle **any type of request (protocol-independent)**.*

   *Usually extended to create **protocol-specific servlets**.*

   Class:** javax.servlet.GenericServlet **Example:

   public class MyServlet extends GenericServlet {

   public void service(ServletRequest req, ServletResponse res) {

   // logic here

   }

   }

2. ***HTTP Servlet:***

   *A subclass of GenericServlet designed specifically for **HTTP requests**.*

   Handles GET, POST, PUT, DELETE methods easily.

   Class:** javax.servlet.http.HttpServlet **Example:

   public class MyHttpServlet extends HttpServlet {

   protected void doGet(HttpServletRequest req, HttpServletResponse res) {

   // logic here

   } }

12.Difference between HTTP Servlet and Generic Servlet.

12.1 Difference Between HttpServlet and GenericServlet

Basis	***GenericServlet***	***HttpServlet***
Package	*javax.servlet*	*javax.servlet.http*
Protocol Support	*Protocol-independent (can handle any type of request).*	*Specifically designed for **HTTP protocol**.*
Class Type	*Abstract class implementing Servlet interface.*	*Subclass of GenericServlet.*

Methods	*Must override service() method.*	*Provides built-in methods like doGet(), doPost(), doPut(), doDelete().*
Ease of Use	*More general, needs extra coding for HTTP-specific tasks.*	*Easier to use for web applications.*
Usage	*Used when protocol is not HTTP.*	*Used in **web applications** using HTTP/HTTPS.*
Example	*public class MyServlet extends GenericServlet*	*public class MyServlet extends HttpServlet*

13.Servlet Life Cycle.

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

1. ***init()*** ○ *Called **once** when the servlet is first loaded into memory.*
   - ○ *Used for **initialization tasks** such as setting up database connections or loading configurations.*
2. ***service()*** ○ *Called **for every client request**.*
   - ○ *Responsible for **processing requests and generating responses**.*
   - ○ *In HttpServlet, it internally calls methods like doGet(), doPost(), doPut(), etc., based on the request type.*
3. ***destroy()*** ○ *Called **once before the servlet is unloaded** from memory.*
   - ○ *Used to **release resources** such as closing database connections or cleaning up memory.*

14.Creating Servlets and Servlet Entry in web.xml

14.1 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() to handle client requests.*

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

public class HelloServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res)
      throws IOException, ServletException {
res.setContentType("text/html");
PrintWriter out = res.getWriter();
      out.println("<h2>Welcome to My First Servlet!</h2>");
   }
}
```

2. Configuring the Servlet in web.xml:

*web.xml (Deployment Descriptor) defines the **servlet mapping** so the web server knows which URL calls which servlet.*

```
<web-app>
   <servlet>
      <servlet-name>hello</servlet-name>
      <servlet-class>HelloServlet</servlet-class>
   </servlet>

   <servlet-mapping>
      <servlet-name>hello</servlet-name>
      <url-pattern>/welcome</url-pattern>
```

</servlet-mapping>

</web-app>

User accesses the URL → http://localhost:8080/appname/welcome

The server looks up /welcome in web.xml.

It finds the mapped servlet HelloServlet and executes its doGet() method.

15.Logical URL and ServletConfig Interface.

15.*1 Explanation of logical URLs and their use in servlets.*

Logical URL:
A logical URL is a user-friendly or virtual name that is mapped to a specific servlet in the web.xml file.
It helps users access a servlet without knowing its actual class name or location.

Purpose / Use:

- ***Makes the URL easier to remember and cleaner.***
- ***Hides the internal servlet class name for security and simplicity.***
- ***Allows changing servlet mappings without modifying the servlet code.***

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>com.example.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>login</servlet-name>

<url-pattern>/userLogin</url-pattern>

</servlet-mapping>

2.Overview of ServletConfig and its methods.

ServletConfig:
ServletConfig is an interface provided by the servlet container to pass initialization parameters and configuration information to a servlet.
It is created by the container for each servlet.

Purpose:

- ***To get initialization parameters defined in web.xml.***
- ***To get the servlet's name and context information.***

Common Methods:

Method	***Description***
getInitParameter(String name)	***Returns the value of a specific initialization parameter.***
getInitParameterNames()	***Returns all initialization parameter names.***
getServletName()	***Returns the name of the servlet.***
getServletContext()	***Returns the ServletContext object (used to share data among servlets).***

16.RequestDispatcher Interface: Forward and Include Methods.

16.1 Explanation of RequestDispatcher and the forward() and include() Methods

RequestDispatcher Interface:
RequestDispatcher is used to forward a request from one servlet (or JSP) to another resource (like another servlet, JSP, or HTML file) within the same server, or to include the content of another resource in the response.

It helps in request chaining and code reusability.

Methods of RequestDispatcher:

1. ***forward(ServletRequest request, ServletResponse response)*** ○

   Forwards the request to another resource.

- ***The control is transferred completely to the new resource.***
- ***The original servlet's output is not shown.*** ○ ***Must be called before response is committed.***

Example:

```
RequestDispatcher rd = request.getRequestDispatcher("welcome.jsp");
rd.forward(request, response);
```

2. ***include(ServletRequest request, ServletResponse response)*** ○ ***Includes the content of another resource in the response.*** ○ ***The control returns to the calling servlet after inclusion.***
   - ○ ***Both servlet outputs are combined.***

Example:

```
RequestDispatcher rd = request.getRequestDispatcher("header.html");
rd.include(request, response);
```

17.ServletContext Interface and Web Application Listener.

17.1 Introduction to ServletContext and its scope.
*ServletContext is an **interface** that provides a way for servlets to **communicate with the web container** and **share information across the entire web application**.*

*It is **common for all servlets** in a web application — created **once** when the application starts and destroyed when it stops.*

Scope:

- *Application-wide (shared by all servlets and JSPs).*
- *Used to store global information like configuration data, database connections, or hit counters.*

Common Methods:

Method	Description
getInitParameter(String name)	Returns the value of a context-wide parameter.
setAttribute(String name, Object value)	Stores data in the application scope.
getAttribute(String name)	Retrieves stored data.
removeAttribute(String name)	Removes an attribute from the context.

Example:

```
ServletContext context = getServletContext(); context.setAttribute("appName",
"StudentPortal");
```

17.2 How to use web application listeners for lifecycle events.

Web Application Listeners:

*Listeners are **special classes** that monitor and respond to **events** in a web application (like when it starts, stops, or when a session is created/destroyed).*

*They are part of the **Servlet API** and help in managing resources automatically.*

Common Listener Types:

Listener	Purpose
ServletContextListener	Detects when the application starts or stops.
HttpSessionListener	Detects creation or destruction of user sessions.
ServletRequestListener	Detects when a request is received or completed.

Example (ServletContextListener):

```
@WebListener
public class AppListener implements ServletContextListener {
public void contextInitialized(ServletContextEvent event) {
```

```
        System.out.println("Application Started");
    }

    public void contextDestroyed(ServletContextEvent event) {
        System.out.println("Application Stopped");
    }
}
```