TOPS Technology

Module – 8 Web Technologies in Java

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

1) Introduction to HTML and its structure

Answer: HTML, or Hyper Text Markup Language, is the standard markup language used to create web pages. It provides the structure and content for a website, allowing users to access and view web pages. HTML consists of a series of elements, represented by tags, which are used to define the different parts of a web page, such as headings, paragraphs, images, and links. The basic structure of an HTML document includes the doctype declaration, the html element, the head element, and the body element.

Basic Structure of HTML:

- <! DOCTYPE html>: Declares the document type as HTML5.
- <Html>: The root element of the HTML document.
- <head>: Contains metadata about the document, such

Structure:

```
<! DOCTYPE html>
<html>
<head>
        <title> -------- </title>
</head>
<body>
        -------------------------
        -------------------------
        -------------------------
</body>
</html>
```

2) Explanation of key tags:

- <a>: Anchor tag for hyperlinks.
- <form>: : Form tag for user input.
- <table>: Table tag for data representation
- <img>: : Image tag for embedding images.
- List tags: <ul>, <ol> and
- <p>: Paragraph tag.
-
: Line break.
- <label>: : Label for form inputs.

Answer:

The **<a> tag** defines a hyperlink, which is used to link from one page to another.

The **<form> tag** is used to create an HTML form for user input. It serves as a container for various input elements like text fields, checkboxes, and buttons, enabling the collection of data for submission to a server or processing via client-side scripts

HTML Tables allow you to arrange data into rows and columns on a web page, making it easy to display information like schedules, statistics, or other structured data in a clear format

<tr> to define table rows,
<th> for table headers, and
<td> for table data cells

The **img tag in HTML** is used to embed an image into a web page. It is a self-closing tag, meaning it does not require a separate closing tag. The basic syntax of the img tag is:

```
<img src="image_url" alt="image_alt_text">
```

In HTML, a list tag is used to define an unordered or ordered list. There are two main types of list tags:

.<ul>: This tag is used to define an unordered list, where items are displayed without any specific order.

: This tag is used to define an ordered list, where items are displayed in a specific order, often with numbers or letters.

Inside these list tags, you can use the <li> tag to define each list item

```
<ul>
<li>Item 1</li>
<li>Item 2</li>
<li>Item 3</li>
</ul>

<ol>
<li>Item 1
```

The <p>tag is commonly used to separate paragraphs of text within a document. It is a self-closing tag, meaning it does not need a separate closing tag.

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

The
 tag in HTML is used to create a line break in an HTML document. It is a self-closing tag, meaning it does not require a closing tag. When used, the
 tag will insert a line break, forcing the content to move to the next line.

```
<p>This is a paragraph of text.<br>This is the next line of text.</p>
```

The **<label> HTML** element represents a caption for a form element in a user interface. It improves accessibility by linking text to form elements. When a user clicks on the **label**, it automatically focuses on or activates the **associated input**, such as text fields, checkboxes, or radio buttons. This helps make forms more **user-friendly** and easier to navigate.

CSS: Inline CSS, Internal CSS, External CSS

1) Overview of CSS and its importance in web design.

Answer: CSS, or Cascading Style Sheets, is a styling language used to control the layout and appearance of web pages written in HTML or XML. It allows developers to separate presentation from content, making it easier to maintain and update websites. CSS is crucial in web design as it enables the creation of visually appealing, responsive, and user-friendly interfaces. With CSS, designers can specify font styles, colors, spacing, and more, ensuring a consistent look and feel across a website. Its importance lies in its ability to enhance user experience and improve website usability.

2) Types of CSS:

- **Inline CSS: Directly in HTML elements.**
- **Internal CSS: Inside a <style> tag in the head section.**
- **External CSS: Linked to an external file**

Answer:

1. Inline CSS

Inline CSS involves applying styles directly to individual HTML elements using the style attribute. This method allows for specific styling of elements within the HTML document, overriding any external or internal styles.

2. Internal or Embedded CSS

Internal or Embedded CSS is defined within the HTML document's <style> element. It applies styles to specified HTML elements. The CSS rule set should be within the HTML file in the head section i.e. the CSS is embedded within the <style> tag inside the head section of the HTML file.

3. External CSS

External CSS contains separate CSS files that contain only style properties with the help of tag attributes (For example class, id, heading, … etc). CSS property is written in a separate file with a .css extension and should be linked to the HTML document using a **link** tag. It means that, for each element, style can be set only once and will be applied across web pages.

CSS: Margin and Padding

1) Definition and difference between margin and padding.

Answer:

Sr.No.	Margin	Padding
1	Margin is the space outside of the component's edge or border.	Padding is the space inside of the component's edge or border.
2	It is possible to set the margin to auto.	It is not possible to set padding to auto.
3	Margin can be of float number or negative.	Padding cannot be negative.
4	Background colour and other styling components do not impact the margin.	Background colour and another styling of a component will affect padding.

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

Answer:

In CSS, margins and padding are two fundamental properties that control the spacing around an HTML element. **Margins** create space outside the element, essentially pushing other elements

away from it. They are the distance between the element and other elements that are not its children. On the other hand, **padding** creates space inside the element, adding a layer of space between the element's content and its border. This space is not visible but takes up space, affecting the element's dimensions and layout. Understanding the difference between margins and padding is essential for effective layout design.

CSS: Pseudo – Class

1) Introduction to CSS pseudo-classes like: hover, :focus, :active, etc

Answer:

CSS pseudo-classes are used to define a specific style for HTML elements when they are in a particular state. Some common pseudo-classes include:

:hover: applies a style when an element is being hovered over with the mouse cursor
:focus: applies a style when an element has received keyboard focus
:active: applies a style when an element is being clicked or pressed
:visited: applies a style when a link has been visited before
:link: applies a style when a link has not been visited before

These pseudo-classes are commonly used to create interactive and dynamic effects on websites.

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

Answer:

Pseudo-classes allow you to style HTML elements based on their state, such as whether they are in a hover state, active state, or focus state. This is achieved using CSS pseudo-class selectors, which are prefixed with a colon (:).

For example,: hover, :active, and :focus are commonly used pseudo-classes. Pseudo-classes can also be used to target elements based on their state in the document, such as: first-child, :last-child, and :nth-child. By using pseudo-classes, you can create interactive and dynamic effects in your web designs without the need for JavaScript.

CSS: ID and Class Selectors

1) Difference between id and class in CSS.

Answer:

In CSS, both id and class are used to style HTML elements, but they serve different purposes.

ID:

Uniquely identifies one element within a document.
Used with the # symbol, e.g., #header.
Can only be used once per document.
Helps with JavaScript targeting and linking to specific elements.

CLASS:

Identifies multiple elements that share a common style.
Used with a dot (.) symbol, e.g., .header.
Can be used multiple times per document.
Allows for more flexibility in styling and reduces code duplication.

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

Answer:

id (Unique Identifier)

Purpose: To uniquely identify a single element within an HTML document.

Usage Scenarios:

1. **JavaScript Manipulation**: Select and manipulate a specific element.
   - Example: document.getElementById("header").style.color = "blue";
2. **Anchor Links**: Navigate to a specific section of the page.
   - Example: Go to Section 1
3. **Form Inputs**: Link labels to form inputs.
   - Example: <label for="username">Username:</label> <input type="text" id="username" name="username">

class (Reusable Class)

Purpose: To apply styles or behaviors to multiple elements.

Usage Scenarios:

1. **Styling Multiple Elements**: Apply the same style to multiple elements.

- Example: .button { background-color: green; }

2. **JavaScript Manipulation**: Add or remove classes to multiple elements.
   - Example: const items = document.getElementsByClassName("highlight");
3. **Responsive Design**: Use media queries to apply different styles.
   - Example: .grid-item { width: 100%; } @media (min-width: 600px) { .grid-item { width: 50%; } }

Introduction to Client-Server Architecture

1) Overview of client-server architecture.

Answer:

A client - server architecture is a type of distributed system that connects multiple clients to a central server. In this model, clients are software applications or devices that request services or resources from the server, which manages and provides access to shared data, services, and applications.

Key Components:
Client: Requests services or resources from the server.

Server: Manages and provides access to shared data, services, and applications.

Communication: Clients and servers communicate through standardized protocols, such as HTTP, FTP, or SSH.

2) Difference between client-side and server-side processing.

Answer:

Client-side scripting	Server-side scripting
Source code is visible to the user.	Source code is not visible to the user because its output of server-sideside is an HTML page.
Its main function is to provide the	Its primary function is to manipulate and provide

Client-side scripting	Server-side scripting
requested output to the end user.	access to the respective database as per the request.
It usually depends on the browser and its version.	In this any server-side technology can be used and it does not depend on the client.
It runs on the user's computer.	It runs on the webserver.
There are many advantages linked with this like faster. response times, a more interactive application.	The primary advantage is its ability to highly customize, response requirements, access rights based on user.
It does not provide security for data.	It provides more security for data.
It is a technique used in web development in which scripts run on the client's browser.	It is a technique that uses scripts on the webserver to produce a response that is customized for each client's request.
HTML, CSS, and javascript are used.	PHP, Python, Java, Ruby are used.
No need of interaction with the server.	It is all about interacting with the servers.
It reduces load on processing unit of the server.	It surge the processing load on the server.

3) Roles of a client, server, and communication protocols.

Answer:

Client:

Initiates requests for services or resources
Sends requests to the server
Receives responses from the server
Typically has a user interface and handles input/output operations

Server:

Hosts and manages resources and services
Receives requests from clients
Processes and responds to client requests
Maintains a persistent connection with clients

Communication Protocols:

Define the rules and standards for communication between clients and servers
Examples include HTTP, FTP, SSH, and SMTP
Enable secure, efficient, and reliable data transfer between clients and servers.

HTTP Protocol Overview with Request and Response Headers

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

Answer:

The Hypertext Transfer Protocol (HTTP) is a fundamental protocol that enables communication between web servers and clients, facilitating the exchange of data and resources over the internet. Introduced in 1991, HTTP is the backbone of the web, responsible for transmitting hypertext documents, images, videos, and other multimedia content. Its primary role is to request and retrieve resources from web servers, using request and response messages to establish and manage connections. HTTP is a stateless protocol, ensuring efficient and scalable communication between servers and clients, making it an essential component of the web's underlying infrastructure.

2) Explanation of HTTP request and response headers

Answer:

Request headers

Sent by the client, these headers contain information about the client and the requested resource. For example, a request message can use headers to indicate its preferred media formats.

Response headers

Sent by the server, these headers include server details such as time, location, and configuration. A response can use headers to indicate the media format of the returned body

---

J2EE Architecture Overview

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

Answer:

Java 2 Platform, Enterprise Edition (J2EE) is a platform designed for building and deploying enterprise-level applications. It provides a set of services, APIs, and protocols that allow the development of multi-tiered, distributed, and web-based applications. J2EE simplifies the development process and reduces the need for writing complex code by offering a suite of standardized, reusable components that streamline application deployment and performance.

Multi-tier Architecture

1. **Client Tier**
    - This is the topmost layer where interaction with users occurs. It could be a web browser, mobile app, or custom client application. The client tier sends requests to the server and displays the results to the user.
2. **Presentation Tier**
    - Also known as the web tier, it handles the presentation logic and the user interface. This tier processes incoming client requests, manages user sessions, and generates responses. Technologies like Servlets, JSP (JavaServer Pages), and frameworks like JSF (JavaServer Faces) are commonly used in this layer.
3. **Business Logic Tier**
    - Sometimes called the application tier, this layer contains the core functional logic of the application. It processes the business rules and workflows. Enterprise JavaBeans (EJBs), which can handle tasks like transaction management, messaging, and security, are often utilized here to facilitate business functions efficiently.
4. **Data Access Tier**
    - Also called the persistence tier, it is responsible for communication with the database. It retrieves data, stores information, and manages database transactions. Technologies like JDBC (Java Database Connectivity), JPA (Java Persistence API), and ORM (Object-Relational Mapping) frameworks interact with the database in this tier.
5. **Integration Tier**

- This optional layer manages interactions with external systems and services. It could involve messaging services, web services, or connectors to legacy systems. Technologies such as Java Messaging Service (JMS), JAX-WS (Java API for XML Web Services), and JAX-RS (Java API for RESTful Web Services) come into play here.

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

Answer:

Web containers, application servers, and database servers have different roles in web applications:

Web containers

Manages the lifecycle of servlets, maps URLs to servlets, and ensures that the requester has the correct access rights. Web containers also provide a run-time environment for web applications and services like database access and security.

Application servers

Delivers virtual applications to end-user devices, and ensures high availability and anywhere access. Application servers also contain a web container and EJB container, and can handle Servlets, JSPs, and EJBs.

Database servers

Part of a typical web application environment that resides on a single server.

Web Component Development in Java (CGI Programming)

1) Introduction to CGI (Common Gateway Interface)

Answer:

CGI stands for common gateway interface. It is an interface that enables the web server to process a user's HTTPS request by executing a program. It executes a program that resides on the server to process data to produce the relevant dynamic content. When a client gives a request or input, this data is passed to the CGI script as an environment variable, and then the CGI script processes the request and generates dynamic content for the user's input. Once the web server receives the output generated by CGI, it sends the response to the client.

2) Process, advantages, and disadvantages of CGI programming.

Answer:

The Common Gateway Interface (CGI) is a set of rules that allows a web server to communicate with other programs, such as external databases, to create dynamic web content. CGI has several advantages and disadvantages, including:

Advantages

Quick implementation: CGI is a simple way to implement basic web tasks, like data manipulation or form processing.

Well-defined standard: CGI is a widely accepted standard that ensures compatibility across different platforms and systems.

Language-independent: CGI scripts can be written in any programming language, such as Python, PHP, Java, or C#.

Supports server - side scripting: CGI can easily interact with databases to access information or modify contents.

Disadvantages

Overhead in page loads: Each page load in CGI incurs overhead by having to load the script into memory.

Security concerns: If not managed correctly, CGI scripts can have security issues, such as exposing sensitive data or allowing unauthorized access.

Scalability issues: CGI struggles with scalability issues, especially when the number of simultaneous requests increases.

Low performance: Every CGI request creates a new process, which can take a long time and use a lot of server resources.

Servlet Programming: Introduction, Advantages, and Disadvantages

1) Introduction to servlets and how they work.

Answer:

A servlet is a Java class that extends the HttpServlet class, which handles HTTP requests and responses. Servlets are server-side programs that run on a web server and interact with a database to provide dynamic content to users. They work by receiving HTTP requests, processing the data, and sending a response back to the client. The servlet container, such as Apache Tomcat, manages the lifecycle of servlets and provides services like request processing, session management, and multithreading. This allows servlets to handle multiple requests concurrently, making them suitable for web applications with high traffic.

2) Advantages and disadvantages compared to other web technologies.

Answer:

Here are some advantages and disadvantages of web applications compared to other web technologies:

Advantages

Web applications can be more cost-effective, easier to update, and more platform independent than other web technologies. They can also be more accessible, easier to develop and maintain, and don't require downloads.

Disadvantages

Web applications can be more dependent on the internet, have reduced speed, and have security concerns. They can also have limited functionality and performance compared to native apps.

Web applications are different from desktop applications in the following ways:

Accessibility: Web applications are accessible from anywhere with an internet connection, while desktop applications need to be installed on each device.

Collaboration: Web applications allow real-time collaboration, while desktop applications don't.

Offline use: Desktop applications can work offline, while web applications can't.

Security: Desktop applications keep data more secure than web applications.

Servlet Versions, Types of Servlets

1) History of servlet versions

Answer:

Servlet API Versions and Features

1.0 (June 1997)

- **Key Features**:
  - Basic support for handling HTTP requests and responses.
  - Primitive servlet lifecycle methods like `init()`, `service()`, and `destroy()`.

2.0 (December 1997)

- **Key Features**:
  - Introduction of servlet chaining (limited support for filtering).
  - Support for session management using `HttpSession`.

2.1 (November 1998)

- **Key Features**:
  - Introduction of the ServletContext for sharing data between servlets.
  - Specification for servlet reusability and scalability.

2.2 (August 1999)

- **Key Features**:
  - Web applications concept introduced with the deployment descriptor (`web.xml`).
  - Enhanced security model.
  - Servlet API separated from the Java Web Server.

2.3 (August 2001)

- **Key Features**:
  - Introduction of filters for request and response interception.
  - Servlet event listeners (`HttpSessionListener` and `ServletContextListener`).
  - Improved session management.

2.4 (November 2003)

- **Key Features**:
  - Alignment with XML Schema for the `web.xml` deployment descriptor.
  - Better integration with JSP (JavaServer Pages).
  - Support for HTTP 1.1 request and response handling.

2.5 (May 2006)

- **Key Features**:
  - Support for annotations, such as `@WebServlet`, `@WebFilter`, and `@WebListener`, reducing the dependency on `web.xml`.
  - Enhanced error handling and exception handling.
  - Support for generics in servlet APIs.

3.0 (December 2009)

- **Key Features**:
  - Major enhancements for ease of development.
  - Asynchronous processing using the `AsyncContext` API.

- Dynamic registration of servlets, filters, and listeners at runtime.
- Improved support for annotations, reducing reliance on deployment descriptors.

3.1 (April 2013)

- **Key Features**:
  - Non-blocking I/O support for better scalability.
  - Support for HTTP protocol upgrades (e.g., WebSocket).
  - Security improvements and additional annotations.

4.0 (August 2017)

- **Key Features**:
  - Support for HTTP/2 protocol, enhancing performance.
  - Improved security with enhancements to `ServletContainerInitializer`.
  - Enhanced annotations for better modularity and dynamic registration.

5.0 (December 2020)

- **Key Features**:
  - Package name change from `javax.servlet` to `jakarta.servlet` due to namespace transition.
  - No new features but focused on renaming and compatibility for future development.

6.0 (May 2022)

- **Key Features**:
  - Enhanced alignment with modern development practices.
  - Further updates for HTTP/2 and HTTP/3 support.
  - Improved integration with newer Jakarta EE specifications.

2) Types of servlets: Generic and HTTP servlets.

Answer:

Servlets are Java-based web components that extend the functionality of a web server. There are two primary types of servlets:

1. **Generic Servlet**: A generic servlet is a servlet that does not extend any particular class and implements the Servlet interface directly. It provides a basic implementation of servlet functionality and can be used as a base class for other servlets.

2. **HTTP Servlet**: An HTTP servlet is a servlet that extends the HttpServlet class and overrides methods such as doGet(), doPost(), etc. to handle HTTP requests. This type of servlet is more commonly used and provides a convenient way to handle HTTP requests and responses.

Difference between HTTP Servlet and Generic Servlet

1) Detailed comparison between HttpServlet and GenericServlet.

Answer:

HttpServlet

Implements the HttpServletResponse and HttpServletRequest interfaces
Handles HTTP requests and responses
Provides methods for handling HTTP request methods (e.g., doGet, doPost)
Suitable for web applications that use HTTP protocols

GenericServlet

Implements the Servlet interface
Handles generic requests and responses
Does not provide methods for handling HTTP request methods
Suitable for web applications that use non-HTTP protocols (e.g., FTP, mail)

Servlet Life Cycle

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

Answer:

The init(), service(), and destroy() methods are part of the servlet life cycle, which is a series of steps a servlet goes through from loading to destruction:

Init()

This method is called once when the servlet is first loaded and receives a ServletConfig object parameter. It creates an instance of the servlet.

service()

This method is called when a client requests something from the servlet. It uses ServletRequest and ServletResponse objects to process the request.

destroy()

This method is called once when the servlet is being taken out of service. It performs tasks like closing database connections, releasing memory, and releasing resources. The servlet container calls this method after all threads in the servlet's service method have exited or after a timeout period has passed.

Creating Servlets and Servlet Entry in web.xml

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

Answer:

1. Create a Servlet Class

1. **Extend the `HttpServlet` class**: Write a class that extends `HttpServlet` and override its lifecycle methods, such as `doGet()` or `doPost()`.
2. **Write servlet logic**: Implement the business logic in these methods to process requests and generate responses.

2. Configure the Servlet in `web.xml`

The `web.xml` file is the deployment descriptor for configuring servlets, filters, and listeners in your web application. It's located in the `WEB-INF` folder of the project.

1. **Define the servlet**: Use the `<servlet>` tag to specify the servlet class.
2. **Map the servlet to a URL pattern**: Use the `<servlet-mapping>` tag to associate the servlet with a specific URL.

Example:-

```
<?xml version="1.0" encoding="UTF-8"?>

<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">



    <!-- Define Servlet -->

    <servlet>

        <servlet-name>HelloWorldServlet</servlet-name>

        <servlet-class>HelloWorldServlet</servlet-class>
```

```
    </servlet>


    <!-- Map Servlet to a URL -->
    <servlet-mapping>
        <servlet-name>HelloWorldServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

Logical URL and ServletConfig Interface

1) Explanation of logical URLs and their use in servlets

Answer:

Logical URLs are a design pattern in servlets that map URLs to business logic. They are used to separate the presentation layer from the business logic layer, making the code more modular and maintainable. A logical URL is typically a shorter, more meaningful URL that corresponds to a specific business operation, such as "newUser" or "login".

In a servlet, logical URLs are typically defined in a configuration file, such as a properties file or a XML file. The servlet container then uses this mapping to route incoming requests to the corresponding business logic. This approach allows for a clean separation of concerns and makes it easier to manage and extend the application.

2) Overview of ServletConfig and its methods.

Answer:

The ServletConfig interface in Java Servlet technology provides configuration information for a servlet. It is used to pass information to a servlet when it is initialized. Here are some key methods of ServletConfig:

getServletContext(): Returns the ServletContext object, which provides a way to interact with the servlet context.

getInitParameter(String paramName): Returns the value of a servlet initialization parameter.

getInitParameterNames(): Returns an enumeration of the servlet initialization parameter names.

getServletContextName(): Returns the name of the servlet context.

RequestDispatcher Interface: Forward and Include Methods

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

Answer:

RequestDispatcher is an interface in Java that receives client requests and sends them to resources like HTML files, JSPs, or servlets. The RequestDispatcher interface has two methods:

forward()

Forwards a request to another resource on the server, such as a servlet, JSP, or HTML file. The forward() method closes the output stream after it's invoked.

include()

Includes the content of a resource in the response. The include() method leaves the output stream open.

ServletContext Interface and Web Application Listener

1) Introduction to ServletContext and its scope.

Answer:

Introduction to ServletContext and its Scope

The ServletContext is a central API in Java Servlet technology that provides a way to store and share data across the entire web application. It serves as a container for shared resources, configuration data, and attributes. The scope of a ServletContext is the entire web application, meaning it is shared across all servlets, filters, and other components within the same application. This allows for efficient sharing of data and reduces the need for redundant code. The ServletContext is typically created when the web application is initialized and destroyed when it is shut down.

2) How to use web application listeners for lifecycle events.

Answer:

Web application listeners monitor specific events in a servlet-based application, such as application startup, session creation, or request handling. They allow you to execute custom logic when these events occur.

Types of Listeners and Uses

1. **ServletContextListener**:
   - **Event**: Application startup/shutdown.
   - **Use**: Initialize or release global resources like database connections.
   - **Example**:

```
public class AppListener implements ServletContextListener {
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("Application started");
    }
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("Application stopped");
    }
}
```

2. **HttpSessionListener**:
   - **Event**: Session creation/destruction.
   - **Use**: Track active sessions or perform user-specific cleanup.
   - **Example**:

```
public class SessionListener implements HttpSessionListener {
    public void sessionCreated(HttpSessionEvent se) {
        System.out.println("Session created");
    }
    public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("Session destroyed");
    }
}
```

3. **ServletRequestListener**:
   - **Event**: Request initialization/destruction.
   - **Use**: Log or track incoming requests.
   - **Example**:

```
public class RequestListener implements ServletRequestListener {
    public void requestInitialized(ServletRequestEvent sre) {
```

```
        System.out.println("Request started");
    }
    public void requestDestroyed(ServletRequestEvent sre) {
        System.out.println("Request ended");
    }
}
```

4. **Attribute Listeners**:
   - **Event**: Attribute addition, removal, or update in application, session, or request scopes.
   - **Use**: React to attribute changes.
   - **Example**:

```
public class SessionAttributeListener implements HttpSessionAttributeListener

{

    public void attributeAdded(HttpSessionBindingEvent event) {
        System.out.println("Attribute added: " + event.getName());
    }
}
```

Configuration

1. **Using Annotations**:
   - Add @WebListener above the listener class.
2. **Using web.xml**:

```
<listener>
    <listener-class>com.example.MyListener</listener-class>
</listener>
```

Submitted By: Ansari Amanhusain K.