Summary of Lifecycle and Object Creation:

- ***Web Application Deployment***
  - When the web application is deployed, the **ServletContext** (holds server information)is created and shared across the entire application.
- ***Servlet Initialization***
  - When a servlet is loaded and initialized, the **ServletConfig** (holds servlet information)object is created to store initialization parameters for that particular servlet.
- ***Handling a Request:***
  - When a client makes a request, the **ServletRequest** and **ServletResponse** objects are created automatically by the server. The servlet processes the request (ServletRequest) and generates a response (ServletResponse).
- ***Methods Execution:***
  - During the request handling, the **getParameter()** method is used to retrieve client-submitted data, **setContentType()** specifies the type of response being sent, and **getWriter()** is used to send the response data back to the client.

RequestDispatcher

RequestDispatcher Interface

- The RequestDispatcher interface is part of the javax.servlet package and is used for resource-to-resource communication within a web application.
- It allows a request to be forwarded to another resource (such as a servlet, HTML page, or JSP) or to include content from another resource in the response.
- It is used to perform below servlet communications
  - Servlet to Servlet Communication
  - Servlet to JSP/HTML Communication

Types of Servlet Communication

We can use RequestDispatcher for two types of Servlet Communications which are listed below,

- Forward Communication
- Include Communication

1) Forward Communication:

- A servlet (ServletProgram1) receives a request from the client.
- Instead of generating a response itself, ServletProgram1 forwards the request to another resource (like ServletProgram2, an HTML, or JSP page).
- The final response is generated entirely by the forwarded resource (ServletProgram2).
- The client only knows about the initial servlet (ServletProgram1), not the forwarded resource.
- After forwarding the request, ServletProgram1 doesn't produce any output, It's often used for redirecting tasks to other components

Method Syntax:

```
public void forward(ServletRequest request, ServletResponse response)
 throws ServletException, IOException;
```

Client --> ServletProgram1 (Request) --> ServletProgram2 (Response)

2) Include Communication:

- ServletProgram1 handles the request and generates part of the response.
- It includes additional content (output) from another resource (like ServletProgram2, an HTML, or JSP page).
- The response from ServletProgram2 is included with the response of ServletProgram1.
- Both the initial servlet (ServletProgram1) and the included resource (ServletProgram2) contribute to the final response.
- It is useful when you need to combine outputs from multiple resources. (client receives **both** the responses)

Method Syntax:

```
public void include(ServletRequest request, ServletResponse response)
 throws ServletException, IOException;
```

Client --> ServletProgram1 (Request & Partial Response)

↓

ServletProgram2 (Included Content)

Q) How to Create a RequestDispatcher Object

- We can create a RequestDispatcher object using the getRequestDispatcher() method from the **ServletRequest** interface. This method takes a path (the relative URL of the resource) as an argument.

```
public RequestDispatcher getRequestDispatcher(String path);
```

Intro on JSP (Java Server Pages)

JSP (Java Server Pages)

- JSP stands for Java Server Pages, which are used to create dynamic web content.
- JSP allows HTML and Java code to be written together.
- JSP files have a .jsp extension.
- It uses various tags to insert Java code into the web page.
- Common tags used in JSP for writing java code are listed below

1) Scripting Tags
2) Directive Tags
3) Action Tags

1) Scripting tags:

Scripting tags are used to embed Java code directly within a JSP page. These tags are categorized into the following:

a) Scriptlet Tag: Used to write Java or Servlet code inside a JSP page.

Syntax: **<%** code **%>**

Example: **<%** int x = 5; **%>**

b) Expression Tag: Used to output values directly onto the web page

Syntax: **<%=** expression **%>**

Example: **<%=** new java.util.Date() **%>**

c) Declaration Tag: Used to declare variables or methods that can be used within the JSP page.

Syntax: **<%!** declaration **%>**

Example: **<%!** int count = 0; **%>**

2) Directive Tags:

Directive tags are instructions for the JSP container, mainly used during the translation phase. These tags are categorized into the following:

a) @page Directive: Specifies page-level settings such as language, content type, and imports

Syntax: **<%@ page attribute="value" %>**

Example: **<%@ page language="java" contentType="text/html" pageEncoding="UTF-8" %>**

b) @include Directive: Includes another file during the translation phase

Syntax: **<%@ include file="filename.jsp" %>**

Example: **<%@ include file="header.jsp" %>**

c) @taglib Directive: Used to import custom tag libraries

Syntax: **<%@ taglib uri="URI" prefix="prefix" %>**

Example: **<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>**

3) Action Tags:

Action tags are used to control the behavior and flow of JSP pages at runtime.These tags are categorized into the following:

a) ***jsp:include:*** Includes a JSP file during request processing

Syntax: **<jsp:include page="filename.jsp" />**

b) ***jsp:forward:***Forwards the request to another resource (JSP, HTML, or Servlet).

Syntax: **<jsp:forward page="nextPage.jsp" />**

c) ***jsp:param:***Used within <jsp:include> or <jsp:forward> to pass parameters to the next page

Syntax: **<jsp:param name="paramName" value="paramValue" />**

d) ***jsp:useBean, jsp:setProperty, jsp:getProperty:*** These tags work together for JavaBean usage in JSP

Syntax: **<jsp:useBean>:** Instantiates or retrieves a bean.

<jsp:setProperty>: Sets bean properties.

<jsp:getProperty>: Gets bean properties.