

SERVLETS AND JSP

Student Guide



NIIT

R190052300201-ARITRA SARKAR

Servlets and JSP

Student Guide

Trademark Acknowledgements

All products are registered trademarks of their respective organizations.
All software is used for educational purposes only.

Servlets and JSP/SG/18-M08-V01
Copyright ©NIIT. All rights reserved.

No part of this publication may be reproduced, stored in retrieval system or transmitted in any form, or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the publisher.

COURSE DESIGN - STUDENT GUIDE

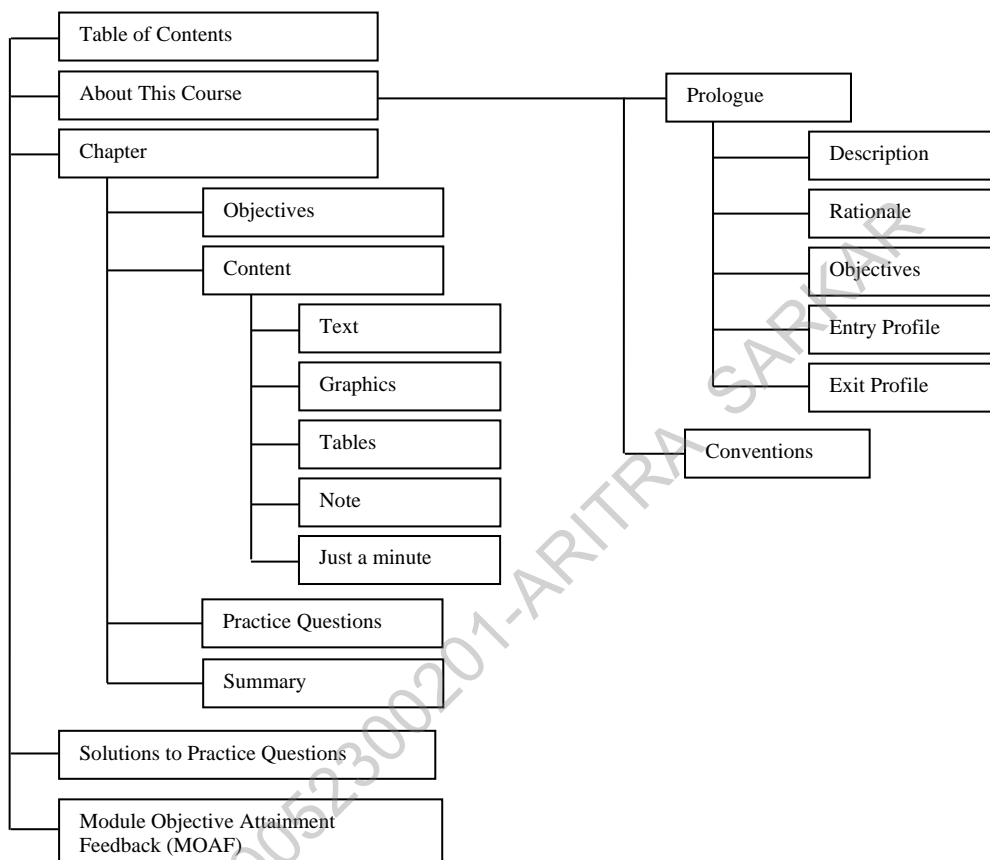


Table of Contents

About This Course

Prologue	ii
Description	ii
Rationale	ii
Objectives.....	ii
Entry Profile	iii
Exit Profile.....	iii
Conventions	iv

Chapter 1 – Introducing Web Application Development

Identify the Various Web Application Technologies	1.3
Practice Questions	1.10
Summary	1.11

Chapter 2 – Exploring the Java Servlet Technology

Introduction to Servlets	2.3
The Servlet API	2.5
Understanding the Web Container.....	2.13
Life Cycle of a Servlet	2.15
Processing of a Servlet Request.....	2.18
Implementing Servlets	2.21
Creating a Servlet.....	2.21
Configuring a Servlet.....	2.22
Compiling and Packaging a Servlet	2.25
Practice Questions	2.27
Summary	2.28

Chapter 3 – Exploring JavaServer Pages Technology

Understanding JSP Technology -----	3.3
Identifying the Components of a JSP Page-----	3.4
Understanding JSP Lifecycle -----	3.17
Lifecycle of a JSP Page-----	3.17
Processing of a JSP Page-----	3.18
Practice Questions -----	3.25
Summary -----	3.26

ABOUT THIS COURSE

R190052300201-ARITRA SARKAR

Prologue

Description

The course, Servlets and JSP, focuses on developing Web applications in the Java EE platform. It introduces Java EE Web components, such as JSP and servlets and how to effectively use these components to develop dynamic Web pages. The course also provides hands-on practices for implementing these techniques.

This course aims at imparting expertise in Web application development using the Web components of Java EE, such as servlets and JavaServer Pages (JSP). In this course, you will learn to create servlets and handle servlet life cycle events.

Rationale

Every organization today needs a global presence in order to expand its business. For the global presence to be felt, the organizations must make their products and services available, according to various time zones, to the customers across the globe. Therefore, the need to make products and services be available globally and continuously led to the development of Web applications.

Web applications have revolutionized the way day-to-day tasks are performed. These applications enable organizations and individuals to share and access information from anywhere, anytime. This has majorly moved the focus of application development from desktop applications to Web applications. Therefore, the tools and technologies required to create interactive Web applications with ease are the demands of the software industry.

Objectives

After completing this course, the students will be able to:

- Identify the Various Web Application Technologies
- Introduce servlets
- Implement servlets
- Understand JSP technology
- Understand JSP lifecycle

Entry Profile

The students who want to take this course should have the knowledge of:

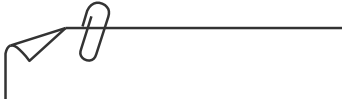
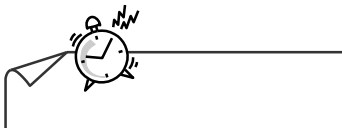

- Core Java
- Java Database Connectivity (JDBC)
- Any RDBMS

Exit Profile

At the end of this course, the students will be able to develop secured Web applications using servlets and JSP.

R190052300201-ARITRA SARKAR

Conventions

<i>Convention</i>	<i>Indicates...</i>
	<i>Note</i>
	<i>Just a minute</i>
	<i>Placeholder for an activity. The steps to complete the activity are included in the book, Servlets and JSP – Lab Guide.</i>



NIIT

R190052300201-ARITRA SARKAR

Introducing Web Application Development

CHAPTER 1

Web applications have revolutionized the way business is conducted. It has enabled the organizations to cater to the need of the customers across the world with great ease. In addition, these applications enable organizations and individuals to share and access information from anywhere and at any time. This has moved the focus of application development from desktop applications to Web applications.

This chapter discusses the various Web application development technologies.

Objectives

In this chapter, you will learn to:

- Identify the Various Web Application Technologies

R190052300201-ARITRA SARKAR

Identify the Various Web Application Technologies

The various Web application development technologies are:

- HTML
- Common Gateway Interface (CGI)
- Java Servlet
- Java Server Pages (JSP)
- ASP.NET
- PHP

HTML

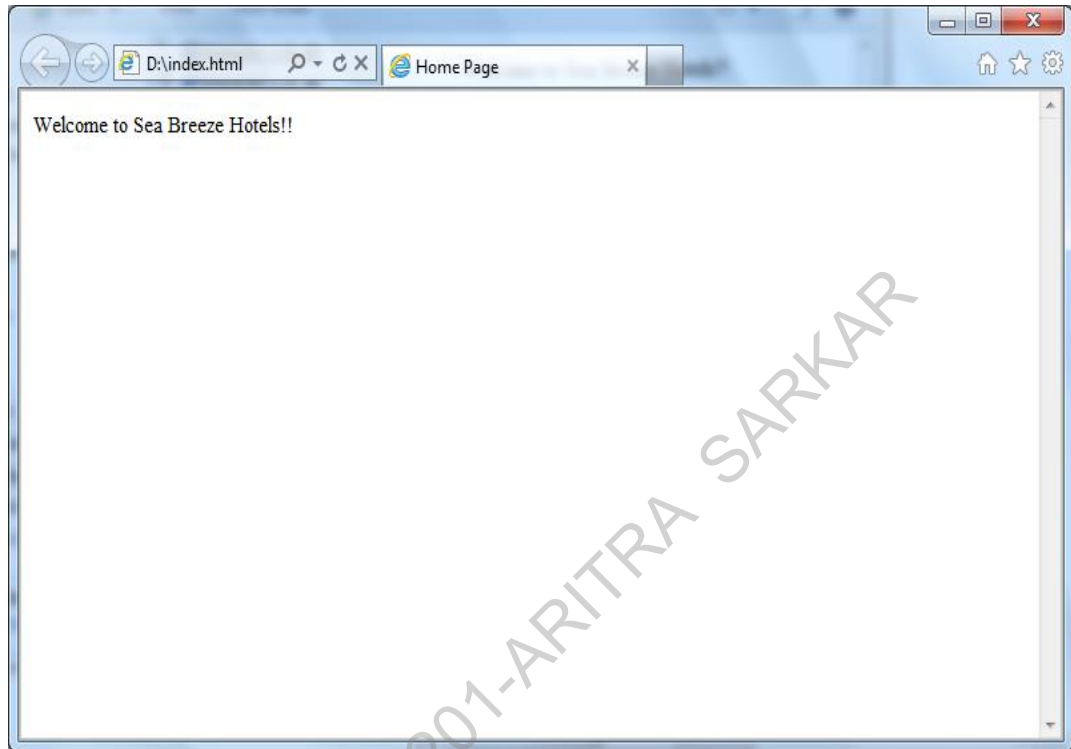
HTML is a versatile markup language that can be used on different Web browsers to publish information on a Web page. A markup language specifies the code for defining, processing, and presenting the text. These codes are called tags.

HTML provides tags that help in creating, formatting (presenting), and enhancing the Web pages by inserting graphics, and specifying fonts and colors for the text. This enhances the visual appeal of the page.

Consider the following code snippet for a basic HTML document:

```
<html>
<head>
<title> Home Page </title>
</head>
<body>
Welcome to Sea Breeze Hotels!!
</body>
</html>
```

The browser interprets the HTML tags to display the Web page. The output of the preceding code is displayed in the following figure.



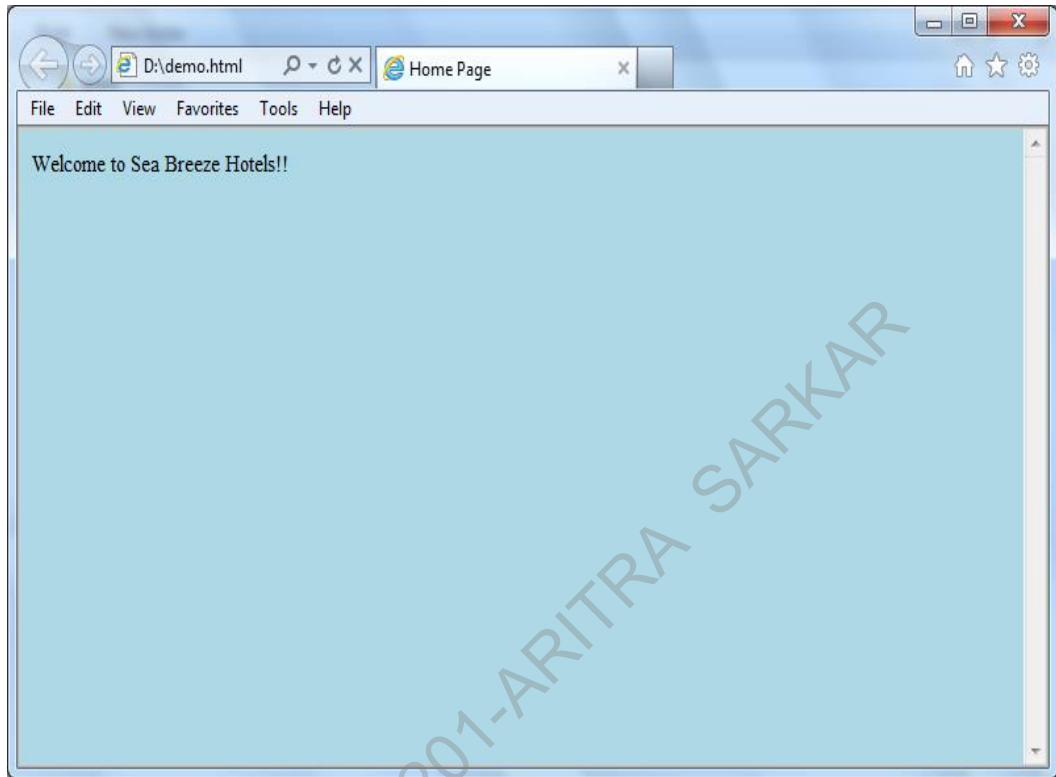
The Code Output Displayed in the Browser

Attributes with their respective values can be used to describe or enhance the tags. The tags process the text or the content as per the specified attributes. For example, a student participates in a painting competition wherein he needs to draw a tree. As per the instructions, the tree has to be a mango tree with the size equal to half of the paper. This scenario can be compared to a markup language. The drawing of the tree represents a tag, and the type and size of the tree are the attributes of this tag.

Consider the following HTML code with an attribute:

```
<html>
<head>
<title> Home Page </title>
</head>
<body bgColor="lightblue">
Welcome to Sea Breeze Hotels!!
</body>
</html>
```

The output of the preceding code is displayed in the following figure.



The Code Output Displayed in the Browser

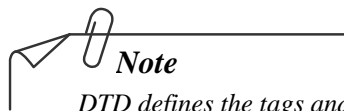
Some of the basic tags of an HTML file are:

- `<!DOCTYPE>`
- `<HTML>`
- `<HEAD>`
- `<BODY>`
- `<SCRIPT>`

<!DOCTYPE> Tag

The `<!DOCTYPE>` tag, represents the Document Type Declaration (DTD). This is the first tag in every HTML document. The Web browser uses the `<!DOCTYPE>` tag information to determine what version of HTML has been used in the document. This is not a mandatory tag. The syntax of the `<!DOCTYPE>` tag is:

```
<!DOCTYPE HTML >
```



Note

DTD defines the tags and attributes that can be used to describe the document.

<HTML> Tag

The <HTML> tag is the first tag after the optional tag <!DOCTYPE> in an HTML document. The opening <HTML> and closing </HTML> tags mark the beginning and end of the Web page, thereby setting the boundary of the HTML document. In a document where you have declared the <!DOCTYPE> tag, use of the <HTML> tag becomes optional, because the Web browser will identify the markup language used in the document as HTML from the DOCTYPE declaration. However, it is always a good practice to declare both the <!DOCTYPE> and <HTML> tags in a document. The syntax of the <HTML> tag is:

```
<HTML>
.....content of the Web page.....
</HTML>
```

<HEAD> Tag

The <HEAD> tag is contained within the <HTML> and </HTML> tags. It is used to describe the header of an HTML document. The header is the first section of an HTML document, where you can define the global settings for the document. A user can download the header information, such as the Web page title, file format, last modified date, and keywords of a document. This information can be used to refer or to generate a resource/information repository for later reference. The syntax of the <HEAD> tag is:

```
<HEAD>
...header content...
</HEAD>
```

The <HEAD> tag consists of the following tags:

- <TITLE> tag
- <BASE> tag
- <LINK> tag
- <META> tag

<TITLE> Tag

The opening <TITLE> and closing </TITLE> tags enclose the title of the document that is displayed in the Web browser's title bar. An HTML document can have only one title. If you do not provide a title, most Web browsers will display the HTML document's name as the default title of the Web page. The syntax of the <TITLE> tag is:

```
<HEAD>
<TITLE>My Home Page</TITLE>
</HEAD>
```


<META> Tag

The <META> tag is an empty tag enclosed within the opening <HTML> and closing </HTML> tags. The <META> tag gives a short description of the contents of the Web page. You can specify the keywords in a Web page by using the <META> tag. This enables search engines to easily classify and identify HTML documents without downloading the entire file. In addition, you can use the <META> tag to provide the author name and the authoring tool used in the document. You can also use the <META> tag to refresh the Web pages automatically.

The syntax for using the <META> tag to provide keywords for search engines is:

```
<META NAME="Keywords" CONTENT="Keyword List">
```

The syntax for using the <META> tag to provide a detailed description of the Web page is:

```
<META NAME="Description" CONTENT="Description for the Web page">
```

An example using the <META> tag to name the author of a Web page is:

```
<META NAME="Author" CONTENT="Michelle Golmes">
```

The syntax for using the <META> tag to refresh a Web page automatically after a specified number of seconds is:

```
<META HTTP-EQUIV="Refresh" Content="Number of seconds"; URL="URL of the Web page">
```

<BODY> Tag

The <BODY> tag sets the boundary for an HTML document. When the Web page is displayed on the Web browser, the users can see the content enclosed within the opening <BODY> and closing </BODY> tags. The <BODY> tags enclose the actual HTML content, including the text, graphics, and forms. The syntax for using the <BODY> tag is:

```
<BODY>
...content...
</BODY>
```

The <BODY> tag mainly consists of <DIV> and tags. The <DIV> tag is a block tag, which divides the structure of the HTML document into different sections. The tag is an inline tag used to mark content inline.

<SCRIPT> Tag

The <SCRIPT> tag specifies the scripting languages used in the Web page, such as VBScript or JavaScript. The <SCRIPT> tag is an optional tag. The content written within the <SCRIPT> tag is displayed as normal text in Web browsers that do not support scripting languages. To avoid this, you should include the <SCRIPT> tag within an HTML comment. This prevents the contents of the <SCRIPT> tag from being displayed in a browser that does not support the JavaScript.

For example, the following code snippet uses the HTML comment tag inside the <SCRIPT> tag:

```
<!--  
<SCRIPT>  
    // Scripting Code.  
</SCRIPT>  
<!--
```

CGI

CGI is a specification for transferring information over the Web between a client and the Web server. CGI specifications are written using C, C++, and Perl programming languages, Perl being the most popular of all. The programs written using CGI specifications are called CGI programs or scripts.

CGI scripts are stored in the Web server and are used to process the client requests. When a client requests for a resource, the Web server instantiates and executes the CGI script to process the request and send a response to the client. However, the same instance does not serve multiple requests of the same client. For every new request a new instance of the CGI script is created. This leads to poor performance of the server when there are multiple concurrent requests.

Java Servlets

Servlets are server-side Java programs that help in developing powerful, efficient, platform-independent, and dynamic Web applications. Unlike CGI scripts, the servlet initialization code is executed only once. Moreover, separate threads in the server handle each client request. This prevents the creation of unnecessary processes, thereby enhancing the performance of the server.

JSP

The JSP technology is a powerful successor of the Servlet technology. JSP is built on top of the Servlet API to rapidly develop server-side and platform independent dynamic Web pages using HTML, XML, and Java code.

A JSP page, after compilation, generates a servlet and, therefore, incorporates all the functionalities of a servlet.

PHP

PHP stands for Hypertext Preprocessor. It is an open-source, server-side programming language used to create dynamic Web applications. It uses HTML and server-side code to perform a specific task. The server-side code starts with the tag, <? and ends with the closing tag, ?>.

ASP.NET

ASP.NET is a server-side technology that enables programmers to create dynamic Web applications. It is built on the Microsoft .NET Framework. Microsoft introduced the .NET Framework to help developers create globally distributed software with Internet functionality and interoperability. It has a number of advanced features, such as simplicity, security, and scalability that help you develop robust Web applications.



Just a minute:

Which one of the following HTML tags specifies the HTML version in an HTML document?

1. `<HTML>`
2. `<!DOCTYPE>`
3. `<HEAD>`
4. `<META>`

Answer:

2. `<!DOCTYPE>`

Practice Questions

1. Which one of the following Web application technologies is primarily used for creating static Web pages of a Web application?
 - a. HTML
 - b. Java Servlets
 - c. JSP
 - d. ASP.NET

R190052300201-ARITRA SARKAR

Summary

In this chapter, you learned that:

- Some of the Web application development technologies are:
 - HTML
 - CGI
 - Java Servlet
 - JSP
 - ASP.NET
 - PHP

R190052300201-ARITRA SARKAR



NIIT

R190052300201-ARITRA SARKAR

Exploring the Java Servlet Technology

CHAPTER 2

With the increase in the usage of the Internet as the information medium, organizations are creating dynamic Web applications. A dynamic Web application consists of dynamic web pages that enable a user to retrieve data from a database, submit data, or fetch some information based on users input. These features can be incorporated by implementing server-side programs. The server-side programs can be written using various technologies, such as Servlet. Servlet is a Java technology that extends all the features of Java. This enables you to develop powerful, dynamic Web applications.

This chapter focuses on the Servlet APIs, lifecycle of a servlet, and servlet request processing mechanism. In addition, it explains creation and deployment of a servlet.

Objectives

In this chapter, you will learn to:

- Introduce servlets
- Implement servlets

Introduction to Servlets

With the increase in number of users accessing the Internet, the Web has evolved greatly. In addition to being a repository of information and a mode of communication, it is now increasingly used as a medium of entertainment, business, and socializing. This has enforced the need of dynamic Web applications. A dynamic Web application responds to the actions performed by a user dynamically.

Consider the Google website that enables you to search documents, links, images, and videos, on any topic. If you want to search only images, you can select the **Images** link on the main page of the website, which redirects you to the page that allows you to search and display only images.

The following steps display graphical depiction of the preceding example:

1. Type **<http://www.google.co.in/>** in the address bar of Internet Explorer. The following page is displayed.



The Main Page of the Google Website

2. Click the **Images** link on the black bar of the page, as shown in the following figure.




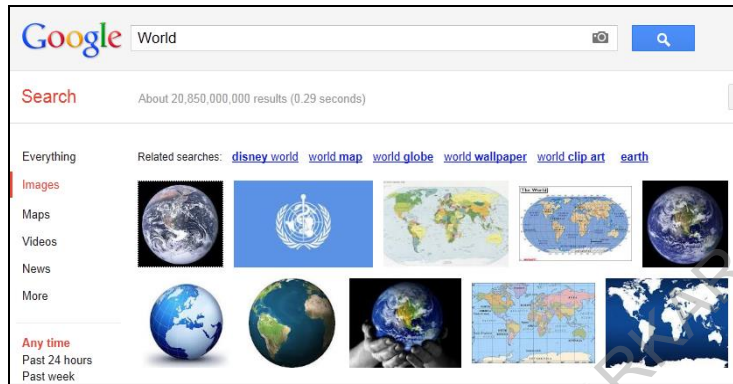
The Home Page of the Google Website

The user is redirected to the search images page, as shown in the following figure.



The Search Images Page

3. Type the word, World, in the search textbox and click the  button. It displays all the images related to the word **World**, as shown in the following figure.



The Sample Result

The preceding example displays the dynamic functionality of the Google website, where the search results are displayed based on the search type and the search text entered by the user. To develop such dynamic Web applications in Java, Servlet technology can be used.

Servlet is a Java program that runs on the server side. It inherits all the features of the Java programming language. In addition, it consists of the following features:

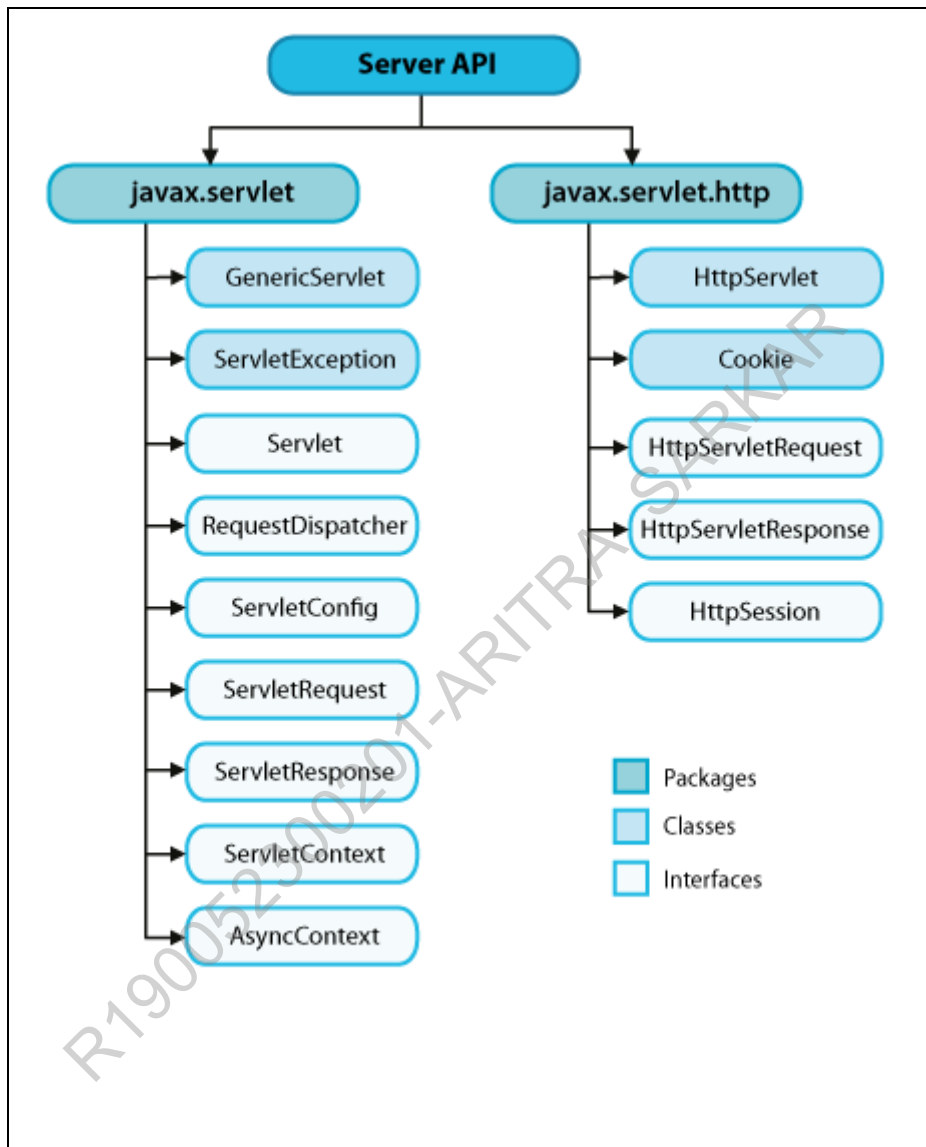
- **Efficient:** Servlets provide faster response to the client requests as it is loaded only once in the Web server memory.
- **Asynchronous support:** Servlet supports asynchronous programming that enables it to serve multiple client requests and generate responses simultaneously. This helps the user to work continuously on the application without being affected by the responses received from the server. In addition, it supports AJAX to implement asynchronous programming that allows a user to interact with the rest of the Web page while the application is processing the changed or updated parts of the Web page.

The Servlet API

To create and manage servlets, various interfaces, classes, and methods are defined in the Servlet API. These classes and interfaces of the Servlet API are available in the following packages:

- `javax.servlet`
- `javax.servlet.http`

The following figure displays the class hierarchy of the Servlet API.



The Servlet API Hierarchy

The javax.servlet Package

The `javax.servlet` package is the integral package of the Servlet API. This package consists of a `Servlet` interface that must be implemented by all servlets.

The following table lists some of the interfaces included in the `javax.servlet` package.

<i>Interfaces</i>	<i>Description</i>
<u>Servlet</u>	<i>It provides the methods that all servlets must implement.</i>
<u>ServletConfig</u>	<i>It provides information to the servlets during the servlet initialization.</i>
<u>ServletContext</u>	<i>It provides methods, which is used by the servlets to communicate with the Web container.</i>
<u>ServletRequest</u>	<i>It provides the client request information to the servlet.</i>
<u>ServletResponse</u>	<i>It helps the servlet in sending the response to the client.</i>
<u>RequestDispatcher</u>	<i>It receives a request from a client and sends it to any other resource, such as a servlet, an HTML page, or a JSP page.</i>
<code>AsyncContext</code>	<i>It provides the methods to handle multiple client requests asynchronously.</i>

The Interfaces of the javax.servlet Package



Note

Servlets run inside a component container in the Web server. This component container is called the Web container, which provides the run-time environment for the execution of servlets.

The javax.servlet.Servlet Interface

The `Servlet` interface provides the methods that must be implemented by all the servlets. To implement these methods, you must create a servlet class that either extends a `GenericServlet` or `HttpServlet` class.

The following table describes the various methods of the `Servlet` interface.

<i>Methods</i>	<i>Description</i>
<code>void init(ServletConfig config)</code> <code>throws ServletException</code>	<i>This method is called by the Web container after creating a servlet instance. This indicates that the servlet is now ready to service the client requests.</i>
<code>ServletConfig</code> <code>getServletConfig()</code>	<i>This method returns a <code>ServletConfig</code> object that contains the configuration information, such as initialization parameters that are passed to a servlet during its initialization.</i>
<code>String getServletInfo()</code>	<i>This method returns a string that contains the information about the servlet, such as author, version, and copyright.</i>
<code>void service(ServletRequest request, ServletResponse response)</code> <code>throws ServletException, IOException</code>	<i>This method is called by the Web container to enable a servlet to generate response for a client request.</i>
<code>void destroy()</code>	<i>This method is called by the Web container just before the servlet instance is taken out of service.</i>

The Methods of the Servlet Interface

The `javax.servlet.ServletConfig` Interface

The `javax.servlet.ServletConfig` interface is implemented by a Web container during the initialization of a servlet to pass the configuration information to a servlet. A servlet is initialized by passing an object of the `ServletConfig` interface to its `init()` method. It is similar to a constructor in a Java class. Therefore, it can be used to pass the initialization parameters to the servlets. The initialization parameters are passed as name-value pairs.

For example, you want that when a Web application is initialized, it should establish a connection with an SQL database named, `BusinessDetails`. In addition, this database is used by most of the pages within the Web application. Therefore, you can specify the connection URL as an initialization parameter of the servlet. When the servlet is initialized, it can use the URL value to obtain a database connection and the same can be shared by the Web pages within the Web application.

The following table lists some of the methods of the `ServletConfig` interface.

<i>Methods</i>	<i>Description</i>
<code>String getInitParameter(String param)</code>	<i>It returns a <code>String</code> object containing the value of the initialization parameters. If the parameter does not exist, it returns <code>null</code>.</i>
<code>Enumeration<String> getInitParameterNames()</code>	<i>It returns the names of all the initialization parameters as an enumeration of <code>String</code> objects. If no initialization parameters have been defined, an empty enumeration is returned.</i>
<code>ServletContext getServletContext()</code>	<i>It returns the <code>ServletContext</code> object for the servlet in which the caller is executing.</i>
<code>String getServletName()</code>	<i>It returns a <code>String</code> object containing the name of the servlet instance.</i>

The Methods of the ServletConfig Interface

The `javax.servlet.ServletContext` Interface

The `ServletContext` interface provides information to all the servlets of a Web application about the environment in which they are running, such as the initialization parameters of the application, the path information of all the files stored in the Web application, and attributes or data that is common to all the servlets.

Each Web application consists of only one `ServletContext` object, also known as a servlet context or Web context. In addition to the information, the `ServletContext` object provides methods that can be used to communicate with the Web container. These methods provide services such as, logging messages to the application server log file or determining the MIME type of a file. To provide the context information, the Web container creates an object of the `ServletContext` interface. This object represents a context within which a servlet executes.

The following table describes the various methods of the `ServletContext` interface.

Methods	Description
<code>public void setAttribute(String, Object)</code>	<i>Binds the object with a name and stores the name-value pair as an attribute of the <code>ServletContext</code> object. If an attribute already exists, this method replaces the existing attribute.</i>
<code>public Object getAttribute(String attrname)</code>	<i>Returns the object stored in the <code>ServletContext</code> object with the name passed as a parameter.</i>
<code>public Enumeration getAttributeNames()</code>	<i>Returns an enumeration of the <code>String</code> object that contains names of all the context attributes.</i>
<code>public String getInitParameter(String pname)</code>	<i>Returns the value of the initialization parameter with the name passed as a parameter.</i>
<code>public Enumeration getInitParameterNames()</code>	<i>Returns an enumeration of the <code>String</code> object that contains names of all the initialization parameters.</i>

The Methods of the `ServletContext` Interface

The `ServletRequest` Interface

The `ServletRequest` interface enables a servlet to handle the client requests by providing the client information to the servlet. For example, a servlet may need to validate the user Id and password entered by a user in the login page of a website. For this purpose, the servlet uses an object of the `ServletRequest` interface to retrieve the user Id and password entered by the user.

The following table describes the various methods of the `ServletRequest` interface.

Methods	Description
<code>public String getParameter(String paramName)</code>	Returns a <code>String</code> object that specifies the value of a particular request parameter.
<code>public String[] getParameterValues(String paramName)</code>	Returns an array of <code>String</code> objects that contains all the values of the request parameter.
<code>public Enumeration getParameterNames()</code>	Returns an <code>Enumeration</code> object containing all the parameter names as <code>String</code> objects that a servlet request contains.
<code>public String getRemoteHost()</code>	Returns a <code>String</code> object that specifies the full-qualified name of the computer from which the request is sent.
<code>public String getRemoteAddr()</code>	Returns a <code>String</code> object that specifies the IP address of the computer from which the request is sent.

The Methods of the ServletRequest Interface

The ServletResponse Interface

The `ServletResponse` interface contains various methods that enable a servlet to respond to the client requests. A servlet can send the response as either character or binary data. The `PrintWriter` stream can be used to send the character data as a servlet response and the `ServletOutputStream` stream can be used to send the binary data as a servlet response.

The following table describes the various methods of the `ServletResponse` interface.

Methods	Description
<code>public ServletOutputStream getOutputStream() throws IOException</code>	Returns an object of the <code>ServletOutputStream</code> class that represents an output stream to send the binary data as response.
<code>public PrintWriter getWriter() throws IOException</code>	Returns an object of the <code>PrintWriter</code> class that the servlet uses to send character data as response.
<code>public void setContentType(String type)</code>	Sets the MIME type for a servlet response. Some of the MIME types are <code>text/plain</code> , <code>image/jpeg</code> , and <code>text/html</code> .

The Methods of the ServletResponse Interface

The javax.servlet.http Package

The `javax.servlet.http` package provides classes and interfaces that enable you to create a servlet that communicates using the HTTP protocol. The following table lists some of the interfaces that belong to the `javax.servlet.http` package.

Interfaces	Description
<code>HttpServletRequest</code>	It extends the <code>ServletRequest</code> interface to represent the request information being sent to the HTTP servlets by an HTTP client.
<code>HttpServletResponse</code>	It extends the <code>ServletResponse</code> interface and provides methods to handle response, status codes, and response headers of the servlets that communicate using HTTP.
<code>HttpSession</code>	It provides a mechanism that helps in identifying a client across multiple requests.

The Interfaces of the javax.servlet.http Package

The `javax.servlet.http` package provides the `HttpServlet` class to create servlets, which communicate using the HTTP protocol. `HttpServlet` is an abstract class that extends from the `GenericServlet` class and provides built-in functionality for the HTTP protocol.

For example, the `HttpServlet` class provides methods, such as `doGet()`, `doPost()`, and `doHead()` that allow a servlet to process a client request coming through a specific HTTP method.

Consider the following code snippet that creates an HTTP servlet named `MyServlet`:

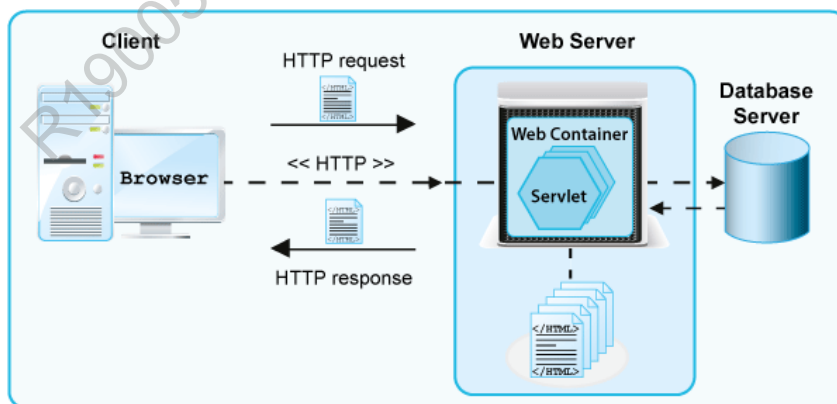
```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet
{
    protected void doGet(HttpServletRequest request, HttpServletResponse
    response)
    {
        ...
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse
    response)
    {
        ...
    }
}
```

Understanding the Web Container

The *Web container* is a component of the Web server which provides the run-time environment for executing the servlets. Whenever a client sends a request to the Web server for a particular servlet, the Web server passes the request to the Web container. The Web container checks whether an instance of the requested servlet exists. If the servlet instance exists, the Web container delegates the request to the servlet, which processes the client request and sends back the response. In case the servlet instance does not exist, the Web container locates and loads the servlet class. The Web container then creates an instance of the servlet and initializes it. The servlet instance, after initialization, starts processing the request. The Web container passes the response generated by the servlet to the client.

The following figure depicts the Web container architecture.

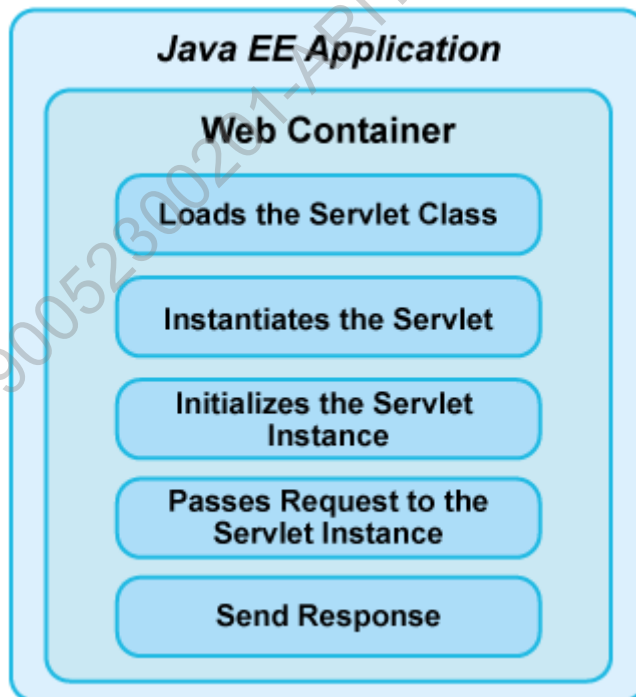


The Web Container Architecture

The Web container implements the Servlet APIs. It provides the following services to the Web applications:

- **Communication Support:** The Web container provides communication support so that a servlet can communicate with the Web server. Therefore, a Web developer only needs to develop the business logic of the servlets.
- **Lifecycle Management:** The Web container manages the lifecycle of the servlets. It manages the loading and instantiation of the servlets. The Web container is also responsible for invoking the servlet methods and making the instance of the servlet eligible for garbage collection.
- **Multithreading Support:** The Web container provides multithreading support for every servlet. Whenever a client requests for a servlet, the container automatically creates a thread and associates the client request with it. When the client request is processed completely, the thread is destroyed by the container.
- **Declarative Security:** The Web container defines the security constraints that specify that only the authorized users can access the deployed servlets. You need to write the complete code in your servlet to make your Web applications secure. Therefore, you can manage and change your security settings without recompiling the Java source files.
- **JSP Support:** The Web container compiles and translates the JSP pages into the servlet class files, which are then used to process the requests of a client.

The following figure shows the tasks performed by the Web container when the client request is received for the first time.



The Tasks Performed by the Web Container

Life Cycle of a Servlet

The lifecycle of a servlet is managed by the Web container. To manage the servlet lifecycle, it uses the methods of the `javax.servlet.Servlet` interface. They are:

- `init()`
- `service()`
- `destroy()`

The `init()` Method

The `init()` method is called during the initialization phase of the servlet life cycle. The Web container first maps the requested URL to the corresponding servlet available in the Web container and then instantiates the servlet. The Web container then creates an object of the `ServletConfig` interface, which contains the startup configuration information, such as initialization parameters of the servlet. The Web container then calls the `init()` method of the servlet and passes the `ServletConfig` object to it.

The `init()` method throws the `ServletException` exception, if the Web container cannot initialize the servlet resources.

The following code snippet shows the `init()` method:

```
public void init (ServletConfig config) throws ServletException
{
...
}
```

Consider a scenario where you want to track the number of users accessing a Web application. To implement this, you need a variable, which will track the number of users requesting the Web application. However, this variable should not be initialized for the subsequent requests. This can be implemented using the following code snippet:

```
public class ServletLifeCycle extends HttpServlet
{
    static int count;

    public void init (ServletConfig config) throws ServletException
    {
        count=0;
    }
}
```

In the preceding code snippet, a variable named `count` is declared inside the `ServletLifeCycle` class. It is initialized with the value 0 in the `init()` method of the servlet class. As the `init()` method is called only once during the lifecycle of a servlet, hence, the `count` variable will be initialized only once.

The service() Method

Once the servlet instance is initialized, it is ready to service the client requests. When the client requests for the servlet, the Web container invokes the `service()` method of the servlet to process the client requests. The Web container passes an object of the `HttpServletRequest` interface and an object of the `HttpServletResponse` interface to the `service()` method. The `HttpServletRequest` object contains information about the request made by the client. The `HttpServletResponse` object contains the information returned by the servlet to the client.

The following code snippet shows the `service()` method:

```
public void service(ServletRequest req, ServletResponse res) throws
ServletException, IOException
{
    ...
}
```

The `service()` method throws the `ServletException` exception when the processing of the client request by the servlet fails. For example, the `service()` method identifies the type of client requests, and dispatches them to the `doGet()` and `doPost()` methods for processing. For example, if the user has clicked on a link, then the `service()` method will call the `doGet()` method.

Therefore, when a client requests for a resource using the HTTP GET method, then the `doGet()` method can be overridden in the servlet class to handle the client request. Similarly, you can override the `doPost()` method in the servlet class to handle the client requests, which are sent by the client using the HTTP POST method.

The destroy() Method

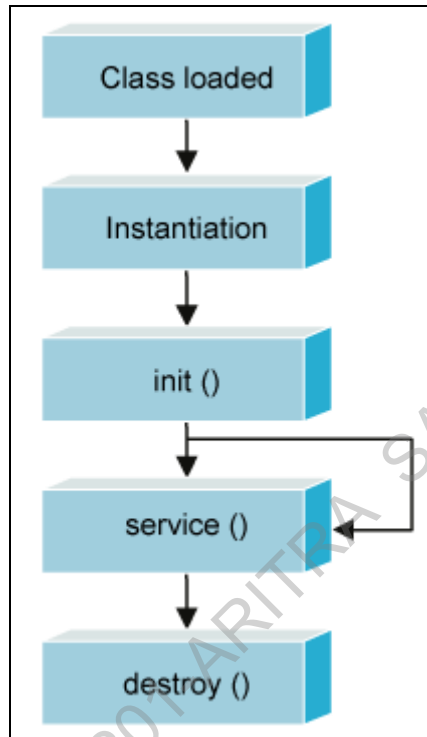
The `destroy()` method marks the end of the life cycle of a servlet. The Web container calls the `destroy()` method when the servlet instance is not required to service any subsequent client request. The Web container calls the `destroy()` method in the following situations:

- The time period specified for the servlet has elapsed. The time period of a servlet is the period for which the servlet is kept in the active state by the Web container to service the client request.
- The Web container needs to release servlet instances to reclaim memory.
- The Web container is about to shut down.
- There are no subsequent requests for a particular servlet instance.

In the `destroy()` method, you can write the code to release the resources occupied by a servlet. In addition, it is used to save any persistent information before the servlet instance is removed from the service. The following code snippet shows the `destroy()` method:

```
public void destroy()
{
}
```

These lifecycle methods are called in a specific sequence by the Web container, as shown in the following figure.



The Servlet Lifecycle Methods Execution Sequence

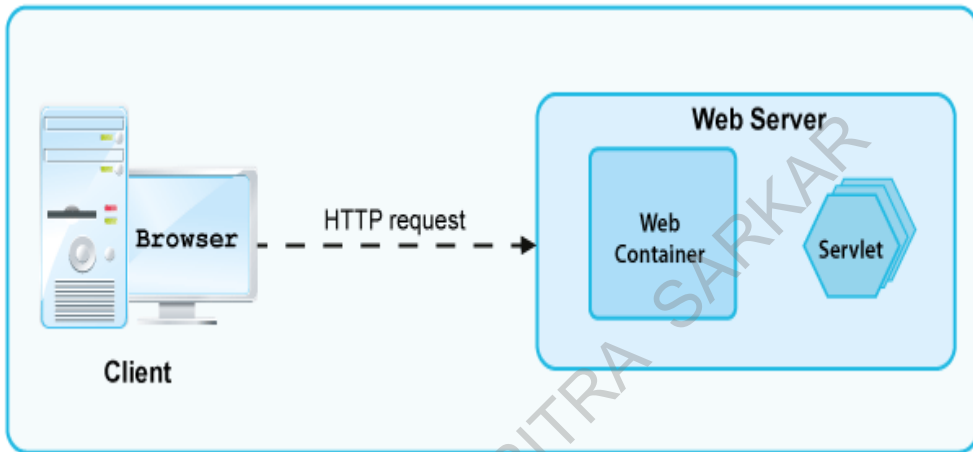
The following is the list of the sequence in which the Web container calls the life cycle methods of a servlet:

1. The Web container loads the servlet class and creates one or more instances of the servlet class.
2. The Web container invokes the `init()` method of the servlet instance during initialization of the servlet. The `init()` method is invoked only once in the servlet life cycle.
3. The Web container invokes the `service()` method to allow a servlet to process a client request.
4. The `service()` method processes the request and returns the response to the Web container.
5. The servlet then waits to receive and process subsequent requests as explained in the steps 3 and 4.
6. The Web container calls the `destroy()` method before removing the servlet instance from the service. The `destroy()` method is also invoked only once in a servlet life cycle.

Processing of a Servlet Request

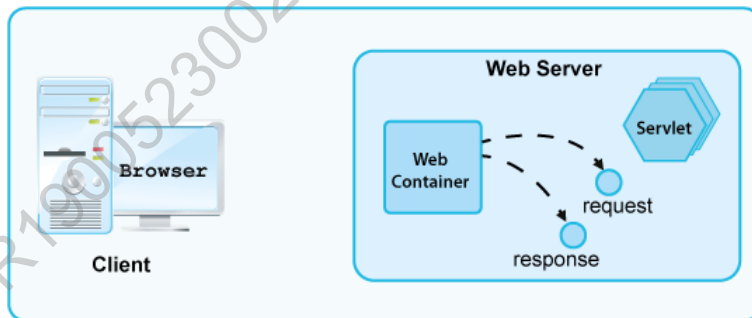
The following steps describe processing of a client request by a servlet:

1. A user sends a request to the server by using the Web browser. The server then forwards the request to the Web container, as shown in the following figure.



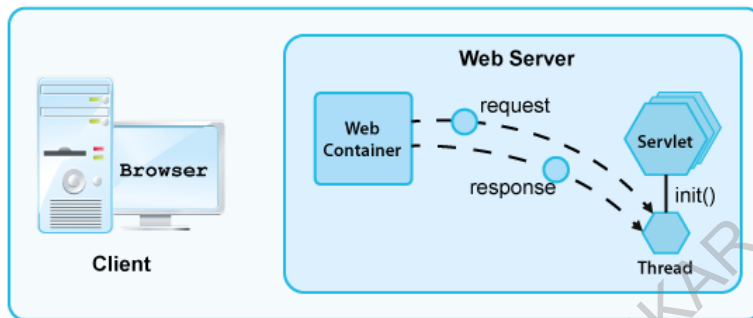
The HTTP Request Being Sent

2. The Web container in turn creates the `HttpServletRequest` and `HttpServletResponse` objects for request handling and response generation, as shown in the following figure.



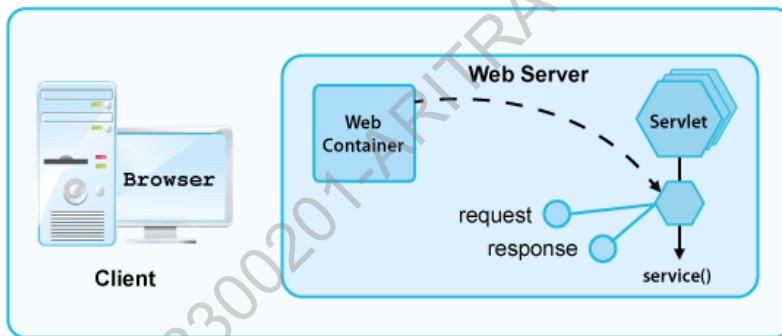
The Creation of Request and Response Objects

3. The Web container identifies the requested servlet, and then initializes the identified servlet thread by calling the `init()` method. In addition, it passes the `HttpServletRequest` and `HttpServletResponse` objects to the initialized servlet thread, as shown in the following figure.



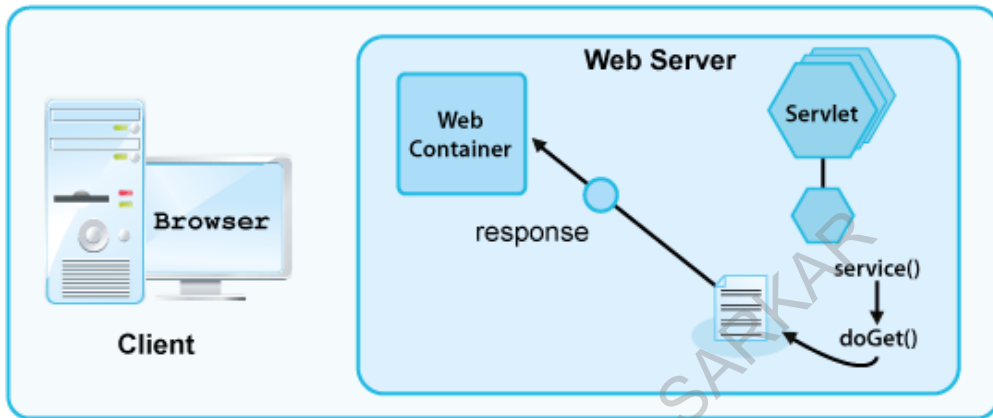
The Creation of a Servlet Thread

4. Now, the Web container calls the `service()` method of the servlet to enable the servlet to process the client requests, as shown in the following figure.



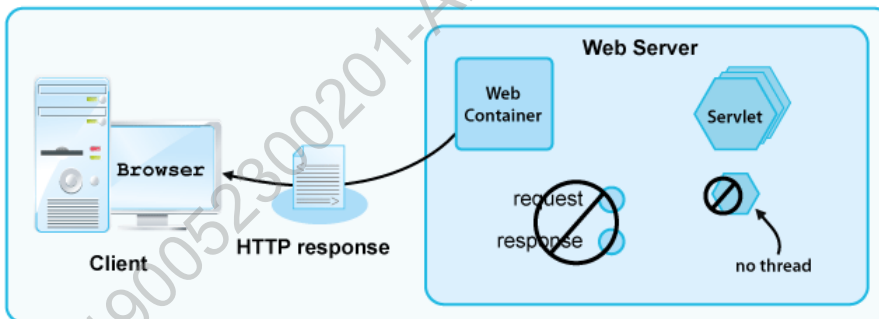
The Invocation of the service() Method

5. Depending upon the type of request, the `service()` method invokes the `doGet()` or `doPost()` method. For example, in the given example if the request is sent to the server by using the GET method, then the `doGet()` method is invoked to service the client, as shown in the following figure.



The Invocation of the doGet() Method

6. The `doGet()` method generates a response for the client and attaches it with the response object. The Web container sends the generated response to the client. Finally, the Web container deletes the servlet instance along with the request and response objects, as shown in the following figure.



The Deletion of Request and Response Objects



The _____ package provides the classes and interfaces to create servlets that support HTTP protocol.

Answer:

`javax.servlet.http`

Implementing Servlets

To create a servlet, you need to perform the following tasks:

1. Create a servlet that either extends the `HttpServlet` or `GenericServlet` class.
2. Configure the servlet.
3. Compile and package the servlet.
4. Deploy the servlet on the application server.
5. Access the servlet using a browser application.

Creating a Servlet

Consider an example, where you have been asked to create a servlet, which displays the current date to the user, each time the servlet is accessed. To implement this, you have to create a servlet that extends from the `HttpServlet` class, as shown in the following code snippet:

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.PrintWriter;

public class CurrentDate extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<h1>" + new java.util.Date() + "</h1>");
        out.println("</BODY></HTML>");
    }
}
```

In the preceding code, the `CurrentDate` class extends the `HttpServlet` class and overrides the `doGet()` method. The `setContentType()` method associated with the `HttpServletResponse` object is used to set the content type of the response to `text/html`. The `getWriter()` method of the `HttpServletResponse` object is used to obtain an instance of the `PrintWriter` class. Finally, the `println()` method of the object of the `PrintWriter` class is used to add the HTML markup to the servlet response. After creating the servlet class, it is then configured to be accessed by the Web container.

Configuring a Servlet

After creating the servlet, it must be associated to a URL so that for a particular request the corresponding servlet is called. To implement this, you need to configure the servlet. Following are the ways to configure a servlet:

- Using the Deployment Descriptor (DD) file
- Using annotations

Configuring a Servlet Using the DD File

The DD file is an XML file named **web.xml**. It contains configuration information, such as the initialization parameters, URL mappings, welcome files, and security constraints of a servlet in which it resides.

The following code shows the content of a sample **web.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" 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">
    <servlet>
        <servlet-name>CurrentDate</servlet-name>
        <servlet-class>mypackage.CurrentDate</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>CurrentDate</servlet-name>
        <url-pattern>/CurrentDate</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

Some of the commonly used elements of the **web.xml** file along with their usage are:

- **<servlet>**: It is the root tag, which specifies the configuration details of all the servlets used in a Web application. It consists of the following nested tags:
 - **<<servlet-name>**: It specifies a logical name of a servlet class.. For example, in the preceding code, the **<servlet-name>** tag specifies a logical servlet name `CurrentDate`, for the servlet class, `mypackage.CurrentDate`.
 - **<servlet-class>**: It specifies the name of the servlet class, which defines the servlet. For example in the preceding code, the **<servlet-class>** tag specifies the name of the servlet class, `mypackage.CurrentDate`.

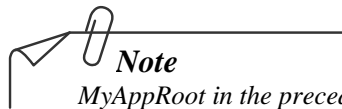
- **<servlet-mapping>**: It consists of the following nested tags:
 - **<servlet-name>**: It contains the same name as specified in the **<servlet-name>** tag nested under the **<servlet>** tag.
 - **<url-pattern>**: It specifies the URL pattern using which a servlet can be accessed. For example in the preceding code snippet, the **<url-pattern>** tag associates the `/CurrentDate` URL pattern to the servlet named `CurrentDate`.

In addition, you can associate a servlet to multiple URL patterns. This is accomplished using multiple **<url-pattern>** tags or by using the wildcard symbol(`*`), as shown in the following code snippet:

```
<servlet-mapping>
<servlet-name>CurrentDate</servlet-name>
<url-pattern>/CurrentDate</url-pattern>
<url-pattern>/MyServlet/*</url-pattern>
</servlet-mapping>
```

The preceding URL patterns will be resolved in the client browser as:

```
http://localhost:8080/MyAppRoot/CurrentServlet
http://localhost:8080/MyAppRoot/MyServlet
http://localhost:8080/MyAppRoot/MyServlet/one
http://localhost:8080/MyAppRoot/MyServlet/twenty
```



Note

MyAppRoot in the preceding URLs refers to the root of the Web application.

- **<session-config>**: It specifies the session information for the servlet, such as the session timeout value, as shown in the following code snippet:

```
<session-config>
  <session-timeout>30</session-timeout>
</session-config>
```

The preceding code snippet specifies that the servlet session will expire after 30 minutes.

In addition to the preceding elements, the **web.xml** file can consist of the following elements:

- **<context-param>**: It is used to specify the servlet context initialization parameters of a Web application. For example, the following code snippet provides the database connectivity information in the initialization parameter:

```
<context-param>
<param-name> JDBC Driver </param-name>
<param-value> Class.forName("oracle.jdbc.driver.OracleDriver");</param-value>
</context-param>
```

- `<init-param>`: It specifies the initialization parameter for a servlet. Unlike the context initialization parameter that is available to all the servlets of a Web application, this parameter is available only to the servlet for which it is declared. The following code snippet shows the usage of the `init-param` element:

```
<init-param>
<param-name> title <param-name>
<param-value> This is the First Servlet </param-value>
</init-param>
```

- `<mime-mapping>`: It specifies the mapping between a file extension and a MIME type, as shown in the following code snippet:

```
<mime-mapping>
<extension>html</extension>
<mime-type> text/html </mime-type>
</mime-mapping>
```

Configuring the servlets by using the **web.xml** file involves extensive coding and is inconvenient during the development process. To simplify this, process annotations can be used.

Configuring a Servlet by Using Annotations

Annotation is a metadata that can be added to the code. It is a keyword that is used to add extra explanation to various programming elements, such as classes, fields, and methods in a program. Annotations are primarily one line code that may further include elements for additional values.

Annotations can also be used to simplify the configuration process of a servlet. The `javax.servlet.annotation` package defines the `@WebServlet` annotation that you can use to configure a servlet.

Consider the following code snippet:

```
@WebServlet(name="CurrentDate",
urlPatterns={"/CurrentDate","/MyServlet/*"})
public class CurrentDate extends HttpServlet
{
    ...
}
```

The preceding code snippet uses the `@WebServlet` annotation to assign the name, `CurrentDate`, to the servlet and associate it to a set of URL patterns, such as `/CurrentDate` and `/MyServlet/*`.

Once the servlet is configured, it needs to be compiled and packaged.

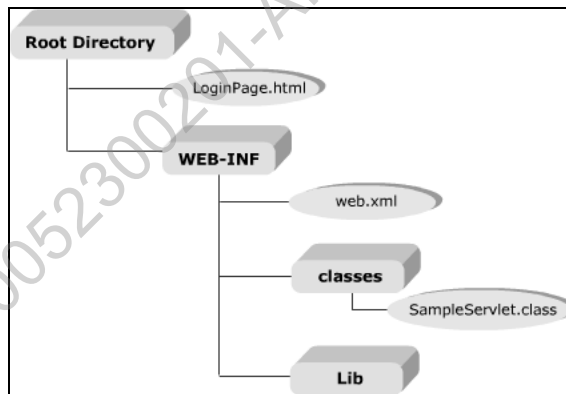
Compiling and Packaging a Servlet

After the servlet is created and configured, it needs to be compiled to generate a servlet class file. While compiling you need to add the classpath of the servlet API using the `-cp` option of the `javac` utility provided in the JDK, if the classpath is not set. After compiling the servlet, you need to package the servlet into the Java EE application structure.

Java EE defines a standard packaging structure to package a servlet into a Java EE application. This makes the Java EE application portable across different application servers. This allows application servers to easily locate and load application files from the standard directory structure. To create the packaging structure of a Web application, you need to create the following directories:

- **The root directory:** The root directory contains static resources, such as HTML files, JSP files, and image files. For example, you can place an HTML file or JSP file in the root directory.
- **The WEB-INF directory inside the root directory:** The **WEB-INF** directory contains the application deployment descriptor file, **web.xml**, which stores various configurations of a Web application.
- **The classes directory:** The **classes** directory present in the **WEB-INF** directory contains the class files of the application. For example, you can place the **SampleServlet.class** file of the `SampleServlet` servlet in the classes directory.
- **The lib directory:** The **lib** directory present in the **WEB-INF** directory contains the Java Archive (JAR) files of libraries that are required by the application components.

The following figure shows the standard packaging structure of a Java Web application.



The Standard Packaging Structure

After you have placed your application specific files in the standard directory structure, you need to package the application into a Web Archive (WAR) file. A WAR file is an archived form of a Java EE Web application. To create a WAR file, type the following command at the command prompt:

```
jar cvf <war filename>
```

This creates the .war file that you can use to deploy the application in the application server.



Activity 2.1: Creating a Servlet



Just a minute:

The _____ contains the static resources, such as HTML files, JSP files, and image files of a Web application.

Answer:

Root directory

Practice Questions

1. Which one of the following features of the servlets enables it to serve multiple client requests and generate responses simultaneously?
 - a. Efficient
 - b. Portable
 - c. Persistent
 - d. Asynchronous support
2. Which one of the following methods is called by the Web container after creating an instance of a servlet?
 - a. `service()`
 - b. `init()`
 - c. `getServletConfig()`
 - d. `getServletInfo()`
3. Which one of the following objects contains the initialization information of a servlet?
 - a. `ServletConfig`
 - b. `ServletContext`
 - c. `ServletRequest`
 - d. `ServletResponse`
4. Which one of the following objects allows interaction of the servlets with the Web container?
 - a. `ServletConfig`
 - b. `ServletContext`
 - c. `ServletRequest`
 - d. `ServletResponse`
5. Which one of the following directories contains the **web.xml** file of the Java Web application?
 - a. `WEB-INF`
 - b. `classes`
 - c. `lib`
 - d. `root`

Summary

In this chapter, you learned that:

- Servlet is a Java program that runs on the server side. It inherits all the features of the Java programming language.
- The Servlet API contains various interfaces, classes, and methods that are used to create and manage servlets. These classes and interfaces are available in the following packages:
 - `javax.servlet`
 - `javax.servlet.http`
- The `Servlet` interface of the `javax.servlet` package defines the methods that the Web container calls to manage the servlet life cycle.
- The `javax.servlet.ServletConfig` interface is implemented by a Web container during the initialization of a servlet to pass the configuration information to the servlet.
- The `javax.servlet.Servlet` interface defines the life cycle methods of a servlet, such as `init()`, `service()`, and `destroy()`.
- The Web container invokes the `init()`, `service()`, and `destroy()` methods of a servlet during its life cycle.
- You can configure a servlet by using the:
 - `web.xml` file.
 - `@WebServlet` annotation.
- To create a servlet class, you need to perform the following tasks:
 - Create a servlet that either extends the `HttpServlet` or `GenericServlet` class.
 - Configure the servlet.
 - Compile the servlet.
 - Package the servlet.
 - Deploy the servlet on the application server.
 - Access the servlet using a browser application.
- To create the packaging structure of a Web application, you need to create the following directories:
 - A root directory
 - A `WEB-INF` directory inside the root directory
 - A `classes` directory
 - A `lib` directory



NIIT

R190052300201-ARITRA SARKAR

Exploring JavaServer Pages Technology

CHAPTER 3

In the Web application development process, the Web designer is responsible for developing the static content of the Web application, such as the page layout. In addition, the Web developer is responsible for developing the dynamic content of the application that includes the presentation or business logic.

However, developing the static content of a Web application using servlets is complex and time consuming. This is because Servlet programming involves extensive coding.

Therefore, identification of the static code content and dynamic code content is required to incorporate any changes. To simplify this process, JSP was introduced. JSP facilitates this by segregating the work of the Web designer and the Web developer.

This chapter identifies the various components of a JSP page. In addition, it explains the lifecycle of a JSP page.

Objectives

In this chapter, you will learn to:

- Understand JSP technology
- Understand JSP lifecycle

Understanding JSP Technology

A typical Web application consists of the presentation logic, which contains the design and the structure, such as the page layout, of a Web page. In addition, it consists of the business logic or the dynamic content, which involves application of business-specific requirements. During the Web applications development, time is often lost in situations where the developer is required to write code for the static content. For example, consider an online website of Grant University. This website allows the students to register by using the registration page, as shown in the following figure.

The screenshot shows the Grant University website's registration page. The header is dark blue with the university's name in gold. A sidebar on the left lists navigation options. The main area features a registration form with various input fields and radio buttons for gender selection. The form is titled 'Registration Form' and includes 'Submit' and 'Reset' buttons at the bottom.

The Grant University Registration Page

As a developer, to create the registration page using servlets, you would need to write many `out.println()` statements. For example, to display the **First Name** label along with the text box field in the registration form, you would have to write the following code snippet in a servlet:

```
out.println("<Table>");
out.println("<Tr>");
out.println("<Td>First Name:");
out.println("</Td>");
out.println("<Td>");
out.println("<input type='text'>");
out.println("</Td>");
out.println("</Tr>");
...
```

Similarly, you will have to write these many lines of code to design and display the other components of the UI. As a developer, this becomes a tedious and time-consuming task. To overcome this limitation and simplify the UI creation of Web applications, JSP technology was introduced.

For example, the preceding code snippet can be simplified by using the following JSP code snippet:

```
<Table>
<Tr>
<Td>First Name:</Td>
<Td><input type="text"></Td>
</Tr>...
```

The preceding JSP code snippet is simple and easy to understand. Therefore, it becomes simple and easy for a Web designer to design and formulate the layout for the Web page by using HTML. In addition, a Web developer can work independently and use Java code and other JSP specific tags to code the business logic. This simultaneous construction of the static and dynamic content facilitates development of quality applications with increased productivity.

Identifying the Components of a JSP Page

A JSP page consists of regular HTML tags representing the static content, and the code enclosed within special tags representing the dynamic content. These tags begin with the “<%” symbol and end with the “%>” symbol.

Typically, a JSP page consists of various components that you can use in your Web page for additional functionality. They are:

- JSP comments
- JSP directives
- JSP declarations
- JSP scriptlets
- JSP expression
- JSP actions
- JSP implicit objects

The following table lists the syntax of some of the components of a JSP page.

<i>Component</i>	<i>Syntax</i>
<i>JSP comments</i>	<code><%-- comment --%></code>
<i>JSP directives</i>	<code><%@ directive %></code>
<i>JSP declarations</i>	<code><%! declaration %></code>
<i>JSP scriptlets</i>	<code><% %></code>
<i>JSP expression</i>	<code><%= expression %></code>

The Components of a JSP Page

JSP Comments

Comments explain the JSP code and make it more readable. It is not compiled and hence, is not included in the HTTP response. A comment can be added to a JSP page in the following ways:

- `<%-- comments --%>`
- `<% /** this is a comment ... */ %>`
- `<!-- comments ... -->`

JSP Directives

JSP directives provide global information about a particular JSP page. To add a directive to the JSP page, you need to use the symbol, `<%`, as the prefix and the symbol, `%>`, as the suffix of the directive name. A directive can have more than one attributes. The syntax for defining a directive is:

```
<%@ directive attribute="value" %>;
```

The various JSP directives are:

- The page directive
- The taglib directive
- The include directive

The page Directive

The page directive defines the attributes that notify the Web container about the general settings of a JSP page. You can specify different attributes with the page directive. The syntax of the page directive is:

```
<%@ page attribute_list %>;
```

The following table describes the various attributes supported by the page directive.

<i>Attribute</i>	<i>Description</i>	<i>Syntax</i>
<i>language</i>	<i>Defines the scripting language of the JSP page.</i>	<code><%@page language="java"%></code>
<i>extends</i>	<i>Defines the parent class that the JSP generated servlet extends.</i>	<code><%@page extends="myapp.Validation"%></code>
<i>import</i>	<i>Imports the list of packages, classes, or interfaces into the generated servlet.</i>	<code><%@page import="java.util.Date"%></code>
<i>session</i>	<i>Specifies if the generated servlet can access the session or not. An implicit object, session, is generated if the value is set to true. The default value of the session attribute is true.</i>	<code><%@page session="false"%></code>

Attribute	Description	Syntax
<i>buffer</i>	<i>Specifies the size of the out buffer. If size is set to none, no buffering is performed. The default value of buffer size is 8 KB.</i>	<code><%@page buffer="10kb"%></code>
<i>autoFlush</i>	<i>Specifies that the out buffer be flushed automatically if the value is set to true. If the value is set to false, an exception is raised when the buffer is full. The default value of the autoFlush attribute is true.</i>	<code><%@page autoFlush = "false"%></code>
<i>isThreadSafe</i>	<i>Specifies whether a JSP page is thread-safe or not. The default value of the isThreadSafe attribute is true.</i>	<code><%@page isThreadSafe = "false"%></code>
<i>errorPage</i>	<i>Specifies the URL of a JSP page that will handle any unchecked Java exception.</i>	<code><%@page errorPage="ErrorPage.jsp"%></code>
<i>isErrorPage</i>	<i>Specifies that the current JSP page is an error page, if the attribute value is set to true. The default value of the isErrorPage attribute is false.</i>	<code><%@page isErrorPage="true"%></code>
<i>isELIgnored</i>	<i>Specifies that the current JSP page will ignore all the EL expressions, if this attribute is set to true. The default value of the isELIgnored attribute is false.</i>	<code><%@page isELIgnored="true"%></code>
<i>info</i>	<i>Provides a description of a JSP page.</i>	<code><%@page info="This JSP page will display the Home Page of Grant University website"%></code>
<i>pageEncoding</i>	<i>Specifies the language used by the JSP page to send the response to the Web browser.</i>	<code><%@page pageEncoding="UTF-8"%></code>

<i>Attribute</i>	<i>Description</i>	<i>Syntax</i>
<i>contentType</i>	<i>Defines the MIME type for a response. The default value of the contentType attribute is text/html.</i>	<pre><%@page contentType="text/html"%></pre>

The Attributes of the page Directive

The following code snippet imports the `java.util.Date` package in a JSP page:

```
<%@page import="java.util.Date" contentType="text/html" pageEncoding="UTF-8"%>
```

The preceding code snippet imports the package, `java.util.Date`, by using the `import` attribute of the page directive.

The following code snippet displays the use of the `errorPage` attribute of the page directive:

```
%@page errorPage="ErrorPage.jsp" contentType="text/html" pageEncoding="UTF-8"%>
```

The preceding code snippet redirects a user to the error page named **ErrorPage.jsp**, whenever there is an unhandled exception in the current page. This is implemented by assigning the name of the error page as a value to the `errorPage` attribute of the page directive.

Now, if you want the **ErrorPage.jsp** page should display the exception message to the user stating the cause of exception, you can use the `isErrorPage` attribute in the page directive of the JSP page, as shown in the following code snippet:

```
<%@page isErrorPage="true" contentType="text/html" pageEncoding="UTF-8"%>
```

The preceding page code snippet sets the `isErrorPage` attribute to `true`. This enables the **Error.jsp** page to use the `getMessage()` method of the `Exception` class to retrieve the exception message. This retrieved message is then displayed in the **ErrorPage.jsp** page.

The include Directive

The include directive includes the output of the file specified using the `file` attribute in the form of its relative URL in the calling JSP page. The file is included in the calling page at the translation time. Therefore, if you have made any changes to the included file, it will be reflected only after the next compilation of the JSP page. The syntax to define the include directive is:

```
<%@ include file = "URLname" %>
```


Consider an example, where for the Grant University Web application you have created a JSP page named University.html, which consists of the logo, as shown in the following figure.



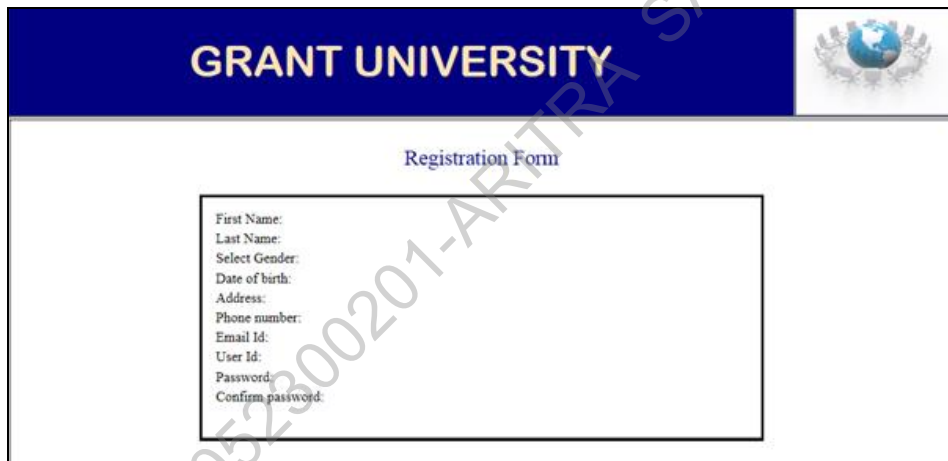
The University.html Page

Now, you want that this logo should be visible in all the other Web pages of the application. This can be achieved either by copying and then pasting the logo code to all the other Web pages or by simply using the include directive in the respective pages.

The following code snippet uses the include directive in the registration page:

```
<%@ include file = "University.html" %>
```

The following figure displays the output of the registration page after using the include directive.

The image shows a web page layout. At the top, there is a dark blue header bar with the text 'GRANT UNIVERSITY' in yellow. To the right of the text is the same globe logo seen in the previous figure. Below the header, the page has a white background. In the center, there is a rectangular box with a thin black border. Inside this box, the text 'Registration Form' is centered at the top. Below it, there is a list of form fields: 'First Name:', 'Last Name:', 'Select Gender:', 'Date of birth:', 'Address:', 'Phone number:', 'Email Id:', 'User Id:', 'Password:', and 'Confirm password:'. Each field is followed by a small, empty text input box. A large, diagonal watermark 'ARITRA SARKAR' is visible across the entire page.

The Registration Page

The taglib Directive

The taglib directive is used to import a custom tag into the current JSP page. Custom tag is a user-defined tag, which is used to perform repetitive tasks in a JSP page. For example, you can create a custom tag that implements the functionality of login controls, which can be used in various Web pages of a Web application. The Tag Library Descriptor (TLD) file defines the functionality of a custom tag.

The taglib directive associates itself with a Uniform Resource Identifier (URI) to uniquely identify a custom tag. It also associates a tag prefix string that will distinguish a custom tag from the other tag library used in the JSP page. The syntax to import a taglib directive in the JSP page is:

```
<%@ taglib uri="tag_lib_URI" prefix="prefix" %>
```

The taglib directive accepts two attributes. The following table lists the two attributes of the taglib directive along with their description.

<i>Attribute</i>	<i>Description</i>
<i>Uri</i>	<i>Locates the tld file of a custom tag.</i>
<i>Prefix</i>	<i>Defines a prefix string to be used for distinguishing a custom tag instance.</i>

The Attributes of the taglib Directive

JSP Declarations

The JSP declarations provide mechanism to define variables and methods in a JSP page. The declarative statements are placed within the `<%!` and `%>` symbols and end with a semicolon.

The following code snippet uses JSP declarations to define variables and methods:

```
<%!  
int i=5;  
int add()  
{  
i=i+5;  
return i;  
}  
%>
```

The preceding code snippet declares a variable `i` of `int` data type and initializes it with the value 5. In addition, the method `add()` is defined, which modifies and returns the value of the `i` variable.

JSP Expressions

JSP expressions are used to directly insert values into the response output. The JSP expressions are evaluated when a user makes an HTTP request. The syntax to include a JSP expression in the JSP page is:

```
<%= expression%>
```

The following code snippet uses JSP expressions to evaluate the value of the expression specified within the `<%=` and `%>` symbols:

```
<h1>The product of 5 and 2 is: <%= (2 * 5) %></h1>
```

The output of the preceding code snippet is shown in the following figure.

The product of 5 and 2 is: 10

The Output of the JSP Expression

JSP Scriptlets

A JSP scriptlet consists of Java code snippets that are enclosed within the `<%` and `%>` symbols. They are executed at the request time. The syntax to include valid Java code in JSP scriptlets is:

```
<% //Java code %>
```

The following code snippet uses JSP scriptlets to include Java code in a JSP page:

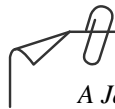
```
<% int i=10;
    if(i>0)
    {
        out.println("i is a positive number");
    }
    else
    {
        out.println("i is a negative number");
    }
%>
```

The preceding code snippet declares a local variable `i` of `int` data type and initializes it with a value `10`. In addition, it displays the message, **i is a positive number** to the users.

JSP Actions

JSP actions are the tags that follow the XML syntax. By using JSP actions you perform tasks, such as inserting files, reusing beans, forwarding a user to another page, and instantiating objects. The following depicts the syntax of JSP actions:

```
<jsp:actionname attribute="">
```



Note

A Java bean is a simple Java class that exposes internal fields as properties using the corresponding getter and setter methods.

The following table lists the various JSP action tags along with their description, attribute supported, and description of attributes.

JSP Action	Description	Attribute	Description of Attribute
<code><jsp:useBean></code>	<i>Invokes and searches for an existing bean.</i>	<i>Id</i> <i>class</i> <i>scope</i> <i>beanName</i>	<i>Uniquely identifies the instance of the bean.</i> <i>Identifies the class from which the bean objects are to be implemented.</i> <i>Defines the scope of the bean.</i> <i>Defines the referential name for the bean.</i>
<code><jsp:getProperty></code>	<i>Retrieves the property of a bean.</i>	<i>name</i> <i>property</i>	<i>Defines the name for the bean.</i> <i>Defines the property from which the values are to be retrieved.</i>
<code><jsp:setProperty></code>	<i>Is used to set the property of a bean.</i>	<i>name</i> <i>property</i> <i>value</i> <i>param</i>	<i>Specifies a name for the bean.</i> <i>Defines the property for which values are to be set.</i> <i>Defines an explicit value for the bean property.</i> <i>Defines the name of the request parameter to be used.</i>
<code><jsp:forward></code>	<i>Is used to forward a request to a target page.</i>	<i>Page</i>	<i>Specifies the URL of the target page.</i>
<code><jsp:include></code>	<i>Includes a file in the current JSP page.</i>	<i>Page</i> <i>flush</i>	<i>Specifies the URL of the resource to be included. The resource is included in the calling page at the run time. Therefore, if you have made any changes to the included resource, it will be reflected in the next request.</i> <i>Specifies whether the buffer should be flushed or not. The flush value can be either true or false.</i>

JSP Action	Description	Attribute	Description of Attribute
<code><jsp:param></code>	<i>Defines a parameter to be passed to an included or forwarded page.</i>	Name value	<i>Defines the name of the reference parameter.</i> <i>Defines the value of the specified parameter.</i>
<code><jsp:plugin></code>	<i>Executes a Java applet or a Java bean.</i>	type code codebase	<i>Defines the type of plug-in to be included.</i> <i>Defines the name of the class to be executed by the plug-in.</i> <i>Defines the path of the code.</i>

The JSP Action Tags

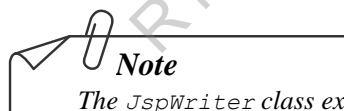
JSP Implicit Objects

JSP implicit objects are predefined objects provided by the container that can be included in JSP expressions and scriptlets. These implicit objects are mapped to the classes and interfaces of the servlet API. The following table describes the various implicit objects of JSP.

Implicit Object	Class	Description
<code>request</code>	<code>javax.servlet.http.HttpServletRequest</code>	<i>It represents the <code>HttpServletRequest</code> object associated with the request.</i>
<code>response</code>	<code>javax.servlet.http.HttpServletResponse</code>	<i>It represents the <code>HttpServletResponse</code> object associated with the response that is sent back to the browser.</i>
<code>out</code>	<code>javax.servlet.jsp.JspWriter</code>	<i>It represents the <code>JspWriter</code> object associated with the output stream of the response.</i>

<i>Implicit Object</i>	<i>Class</i>	<i>Description</i>
<code>session</code>	<code>javax.servlet.http.HttpSession</code>	<i>It represents the <code>HttpSession</code> object associated with the session for the given user of the request. This variable does not exist if JSP is not participating in the session.</i>
<code>application</code>	<code>javax.servlet.ServletContext</code>	<i>It represents the <code>ServletContext</code> object for the Web application.</i>
<code>config</code>	<code>javax.servlet.ServletConfig</code>	<i>It represents the <code>ServletConfig</code> object associated with the servlet for the JSP page.</i>
<code>page</code>	<code>java.lang.Object</code>	<i>It represents the current instance of the JSP page that, in turn, is used to refer to the current instance of the generated servlet.</i>
<code>pageContext</code>	<code>javax.servlet.jsp.PageContext</code>	<i>It represents the page context for a JSP page.</i>
<code>exception</code>	<code>java.lang.Throwable</code>	<i>It represents the <code>Throwable</code> exception in a JSP page.</i>

The JSP Implicit Objects



Note

The `JspWriter` class extends from the `java.io.Writer` class and is used to render output to a JSP page. The object of the `JspWriter` class is referenced by the implicit variable, `out`.

The JSP implicit objects, such as request and response can be used to retrieve the request data from the client and send a response to the client. For example, you have created an application, which accepts and validates the user name and password entered by a user on the login page.

- **UserInput.jsp:** Displays the interface that allows the user to enter the user name and the password. These input values are passed as request parameters to the Welcome.jsp page.
- **Welcome.jsp:** Processes the input given by the user and displays a customized welcome message to the user.

[illegible]

MY FORM

Enter the user name:

Enter the password:

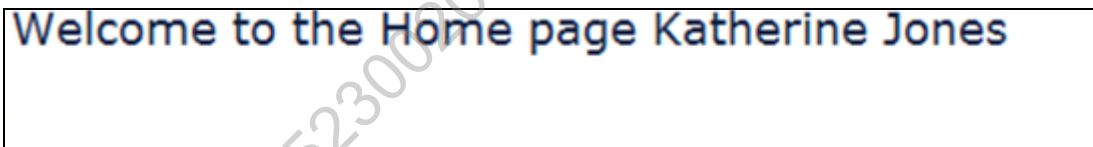
SUBMIT

When a user types **Katherine Jones** and **password@123** in the first and second text boxes, respectively, and then clicks the **SUBMIT** button, the **Welcome.jsp** page is invoked. This page retrieves and validates the value stored in the user name and password fields of the **UserInput.jsp** page and generates a customized welcome message to the user.

The **Welcome.jsp** page consists of the following code:

```
<%-- Import the Java packages --%>
<%@ page language="java"%>
<%@ page import="java.lang.*"%>
<html>
<font size=4 face="Verdana" color=#112244>
<body>
<%
    String str1=request.getParameter("t1");
    String str2=request.getParameter("t2");
    if(str1.equals("Katherine Jones") && str2.equals("password@123"))
    {
        out.println("Welcome to the Home page " + str1);
    }
    else
}
%>
<jsp:forward page="ErrorPage.jsp"></jsp:forward>
<%
}
%>
</font>
</body>
</html>
```

In the preceding code, the user name and password are retrieved using the `getParameter()` method of the `request` object. If the user name and password entered in the **UserInput.jsp** page is valid, a welcome message is displayed to the user, as shown in the following figure.



>Welcome to the Home page Katherine Jones

The Output of the Welcome.jsp Page

Otherwise, the user is forwarded to the **ErrorPage.jsp** page, as shown in the following figure.



Invalid Login!!

The Output of the ErrorPage.jsp Page

The **ErrorPage.jsp** page consists of the following code:

```
<%@ page language="java"%>
<%@ page import="java.lang.*"%>

<html>
<head>
<title>Error Page</title>
```



```
</head>
<body>
    <font color='red' size='+2'>Invalid Login!!
</body>
</html>
```



Just a minute:

Which one of the following components of the JSP page consists of valid Java code snippets that are enclosed within the `<%` and `%>` symbols?

1. JSP scriptlets
2. JSP declarations
3. JSP expression
4. JSP implicit objects

Answer:

1. JSP scriptlets



Understanding JSP Lifecycle

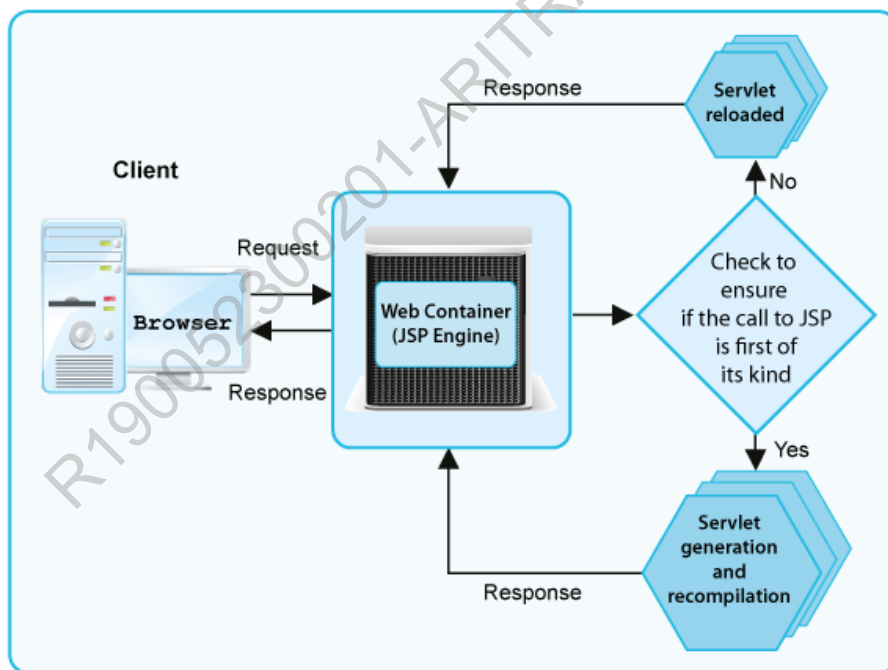
When a client requests for a JSP page, the corresponding JSP page goes through a lifecycle. Similar to servlets, the lifecycle of the JSP page is managed by the Web container. The lifecycle of a JSP page starts when the Web container receives the request for the page and continues for the subsequent request-response cycle of the same JSP page. It ends when no more requests are received for the same JSP page. The JSP page lifecycle is carried out in two phases. They are:

- Translation phase
- Request-processing phase

Lifecycle of a JSP Page

Whenever the client browser requests for a particular JSP page, the server, in turn, sends a request to the JSP engine. A JSP engine is a part of a Web container that compiles a JSP page to a servlet.

The following figure represents the process of the flow of events that occur after a client requests for a JSP page.



The JSP Lifecycle

The Web container determines whether the request for the JSP page is first of its kind. If the JSP page is requested for the first time, it is translated to a servlet. The Web container compiles and executes the servlet to generate the response for the request.

However, if the request for the JSP page is not first of its kind, the corresponding compiled servlet is reloaded to generate the response. The generated response is send to the client.

The lifecycle of a JSP page is managed using the following lifecycle methods of the `javax.servlet.jsp.JspPage` interface:

- `jspInit()`: This method initializes the servlet and is invoked when the corresponding servlet of the requested JSP page is loaded in the Web container for the first time.
- `_jspService()`: This method is invoked when the servlet corresponding to the requested JSP page is initialized and is ready to process the client requests. The Web container invokes this method to execute the initialized servlet to process the JSP page request.
- `jspDestroy()`: This method is invoked by the Web container when the corresponding servlet of the JSP page is not required to service any subsequent requests and needs to be removed from the Web container's memory.

Processing of a JSP Page

A JSP page needs to be converted to a servlet before it can service a client request. The processing of a JSP page is carried out in the following phases:

- **Translation:** Is responsible for translating the JSP code to the servlet code.
- **Compilation:** Is responsible for the compilation of the servlet code to the servlet/bytecode class.
- **Servlet class loading:** Is responsible for loading of the servlet class in the Web container.
- **Servlet instance creation:** Is responsible for creating an instance of the loaded servlet.
- **Servlet initialization:** Is responsible for initializing the servlet instance by calling the `jspinit()` method.
- **Servicing client requests:** Is responsible for servicing the client requests by calling the `jspservice()` method.
- **Servlet destruction:** Is responsible for destroying the servlet by calling the `jspdestroy()` method.

Consider a Web application, which consists of a **Hello.jsp** page. This page displays a hello message to the user along with the number of users accessing the Web application.

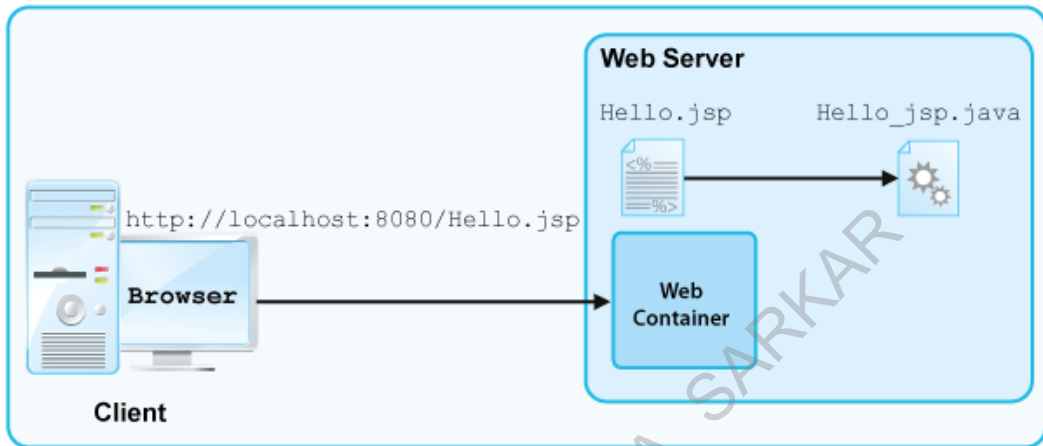
The following code snippet implements this:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <body>
    <%! int i=0; %>
    Hello user number:
    <%=++i %>
  </body>
</html>
```

Whenever a client requests for the **Hello.jsp** page, the processing of the JSP page occurs to process the client request.

Translation

When a client requests for the **Hello.jsp** page, the Web container translates the **Hello.jsp** page to **Hello_jsp.java** servlet, as shown in the following figure.



The Translation of the JSP Code to the Servlet Code

For the **Hello.jsp** page, the Web container creates a servlet named `Hello_Jsp` that extends the `HttpServlet` class, as shown in the following code snippet:

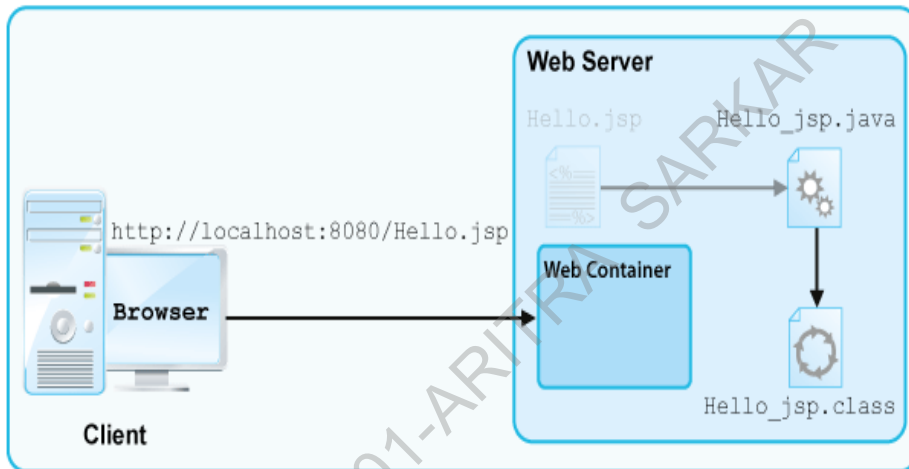
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Hello_Jsp extends HttpServlet
{
    int i=0;
    public void _jspService(HttpServletRequest request, HttpServletResponse
    response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("Hello user number:");
        out.println(++i);
        out.println("</body>");
        out.println("</html>");
    }
}
```

In the preceding code snippet, the `<%! int i=0; %>` declarative expression in the **Hello.jsp** page is translated to the global declaration of variable, `i`, just after the `Hello_JspServlet` class declaration. The HTML tags of the **Hello.jsp** page is translated to the `out.println()` statements in the `_jspService()` method. In addition, the `<%=++i %>` expression statement of the **Hello.jsp** page is translated to `out.println(++i)` statement in the `_jspService()` method.

Compilation

The Web container compiles the **Hello_jsp.java** file to bytecode. This results in the creation of the **Hello_jsp.class** file, as shown in the following figure.

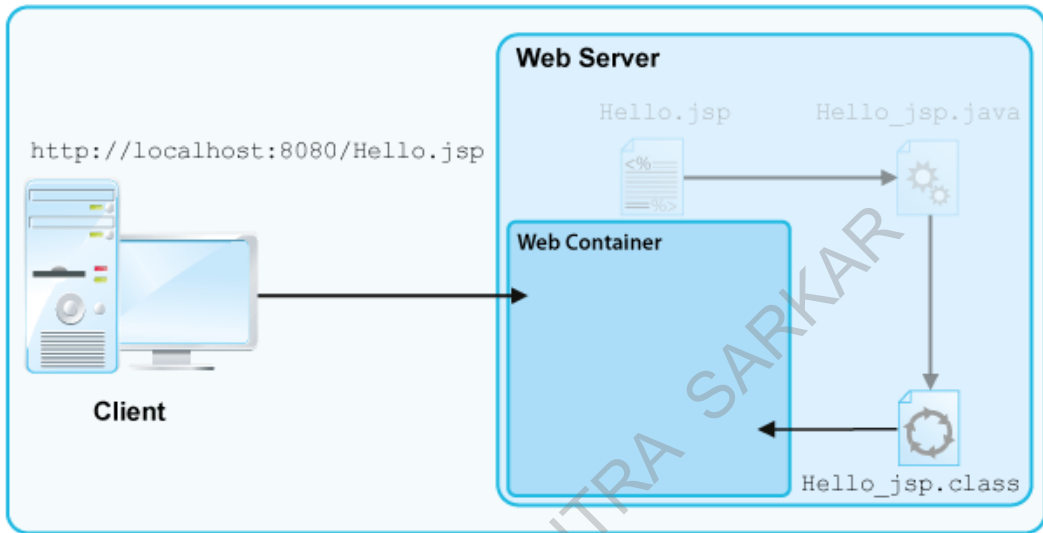


The JSP Page Compilation

Once the code has been compiled, it is ready for execution.

The Servlet Class Loading

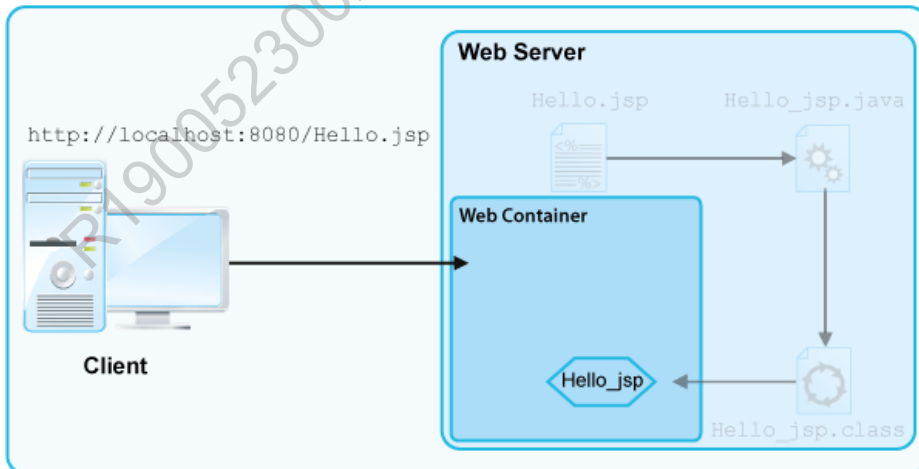
The class loader of the Web container loads the **Hello_jsp.class** file in the Web container's memory, as shown in the following figure.



The Servlet Class Loading

Creation of a Servlet Instance

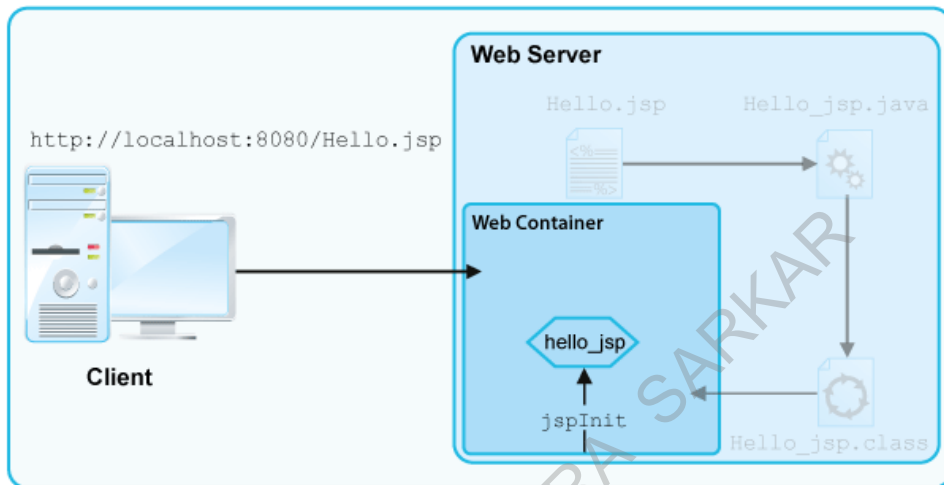
The Web container creates an instance of the **Hello_jsp.class** file, as shown in the following figure.



The Servlet Instance Creation

Servlet Initialization

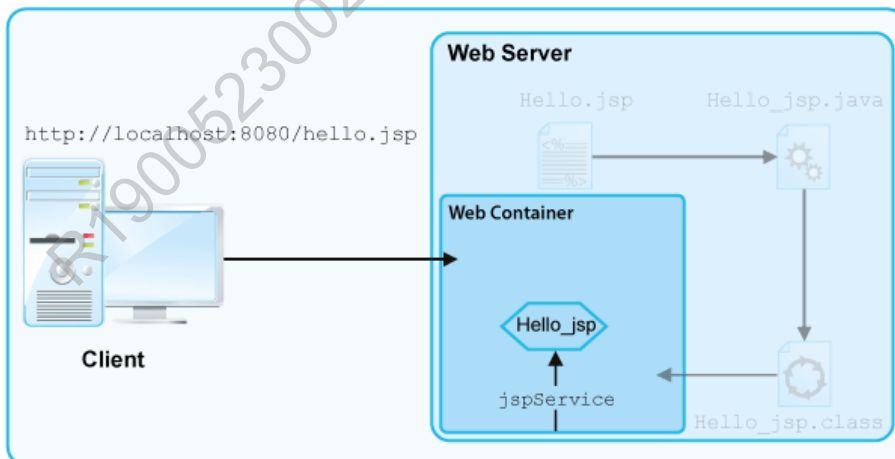
The Web container initializes the instance of the **Hello_jsp.class** file by calling the `jspinit()` method, as shown in the following figure.



The Servlet Initialization

Servicing Client Requests

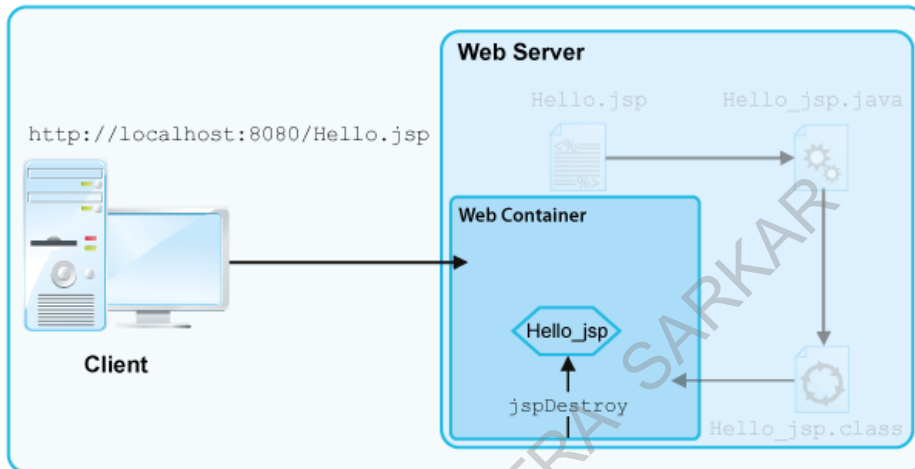
The Web container responds to the client's requests by calling the `_jspService()` method of the initialized `Hello_jsp.class` file, as shown in the following figure.



Servicing the Client Requests

Servlet Destruction

When the servlet is not required to service any client requests, the Web container removes the JSP servlet instance from the Web container's memory by calling the `jspDestroy()` method. This method performs the required clean up. The following figure shows the servlet destruction.



The Servlet Destruction



Activity 3.1: Creating JSP Pages



Just a minute:

Which of the following steps are performed in the translation phase:

- a. Translation of the JSP code to the servlet code.
- b. Compilation of the servlet code to the servlet/bytecode class.
- c. Loading of the servlet class in the Web container.
- d. Creation of the servlet instance.
- e. Initialization of the servlet instance.

- 1. a and b only
- 2. a, b, and c only
- 3. a, b, c, and d only
- 4. c, d, and e only

Answer:

- 1. a and b only



Practice Questions

1. Which one of the following components of a JSP page provides global information about the JSP page?
 - a. JSP directives
 - b. JSP scriptlets
 - c. JSP declarations
 - d. JSP action tags
2. Which one of the following attributes of the page directive specifies that the current JSP page is an error page?
 - a. `isErrorPage`
 - b. `autoflush`
 - c. `isThreadSafe`
 - d. `errorPage`
3. The `Out` implicit object represents the _____ object associated with the output stream of the response.
 - a. `JspWriter`
 - b. `HttpRequest`
 - c. `HttpResponse`
 - d. `ServletConfig`
4. Which one of the following methods is invoked when the corresponding servlet of the requested JSP page is loaded in the Web container for the first time?
 - a. `init()`
 - b. `service()`
 - c. `jspInit()`
 - d. `jspService()`
5. Which one of the following methods is invoked when the servlet corresponding to a JSP page has been initialized and is ready to process the client requests?
 - a. `init()`
 - b. `service()`
 - c. `jspInit()`
 - d. `jspService()`

Summary

In this chapter, you learned that:

- JSP segregates the work of the Web designer and the Web developer.
- The various components of a JSP page are:
 - JSP comments
 - JSP directives
 - JSP declarations
 - JSP scriptlets
 - JSP expression
 - JSP actions
 - JSP implicit objects
- JSP comments are used to add comments in a JSP page to explain the Java code and make the code more readable.
- A directive element in a JSP page provides global information about a particular JSP page. It can be of the following types:
 - The page directive
 - The taglib directive
 - The include directive
- The JSP declarations provide mechanism to define variables and methods in a JSP page.
- JSP expressions are used to directly insert values into the response output.
- JSP scriptlet consists of valid Java code snippets that are enclosed within the `<%` and `%>` symbols.
- JSP action tags are used to include or forward a request to another resource, such as an html page, a servlet, or a JSP page.
- JSP implicit objects are predefined objects that can be included in the JSP expressions and scriptlets. These implicit objects are implemented from the servlet classes and interfaces.
- The JSP lifecycle is managed using the following lifecycle methods of the `javax.servlet.jsp.JspPage` interface:
 - `jspInit()`
 - `jspService()`
 - `jspDestroy()`
- The processing of a JSP page consists of the following two phases:
 - The translation
 - The request processing

Solutions to Practice Questions

Chapter 1

1. a. HTML

Chapter 2

1. d. Asynchronous support
2. b. `init()`
3. a. `ServletConfig`
4. b. `ServletContext`
5. a. WEB-INF

Chapter 3

1. a. JSP directives
2. a. `isErrorPage`
3. a. `JspWriter`
4. c. `jspInit()`
5. d. `jspService()`

Objectives Attainment Feedback

Servlets and JSP

Name: _____ Batch: _____ Date: _____

The objectives of this course are listed below. Please tick whether the objectives were achieved by you or not. Calculate the percentage at the end, fill in your name and batch details, and return the form to your coordinator.

S. No.	Objectives	Yes	No
1.	Identify the Various Web application architecture technologies	<input type="checkbox"/>	<input type="checkbox"/>
2.	Introduce servlets	<input type="checkbox"/>	<input type="checkbox"/>
3.	Implement servlets	<input type="checkbox"/>	<input type="checkbox"/>
4.	Understand JSP technology	<input type="checkbox"/>	<input type="checkbox"/>
5.	Understand JSP lifecycle	<input type="checkbox"/>	<input type="checkbox"/>
Percentage: (# of Yes/5) * 100		<hr/>	

R190052300201-ARITRA SARKAR

NIIT