

Web Component Development Using Java

Web Component Development Using Java Trainer's Guide

© 2015 Aptech Limited

All rights reserved.

No part of this book may be reproduced or copied in any form or by any means – graphic, electronic or mechanical, including photocopying, recording, taping, or storing in information retrieval system or sent or transferred without the prior written permission of copyright owner Aptech Limited.

All trademarks acknowledged.

APTECH LIMITED

Contact E-mail: ov-support@onlinevarsity.com

First Edition - 2015



Dear Learner,

We congratulate you on your decision to pursue an Aptech course.

Aptech Ltd. designs its courses using a sound instructional design model – from conceptualization to execution, incorporating the following key aspects:

- Scanning the user system and needs assessment

Needs assessment is carried out to find the educational and training needs of the learner.

Technology trends are regularly scanned and tracked by core teams at Aptech Ltd. TAG* analyzes these on a monthly basis to understand the emerging technology training needs for the Industry.

An annual Industry Recruitment Profile Survey# is conducted during August - October to understand the technologies that Industries would be adapting in the next 2 to 3 years. An analysis of these trends & recruitment needs is then carried out to understand the skill requirements for different roles & career opportunities.

The skill requirements are then mapped with the learner profile (user system) to derive the Learning objectives for the different roles.

- Needs analysis and design of curriculum

The Learning objectives are then analyzed and translated into learning tasks. Each learning task or activity is analyzed in terms of knowledge, skills and attitudes that are required to perform that task. Teachers and domain experts do this jointly. These are then grouped in clusters to form the subjects to be covered by the curriculum.

In addition, the society, the teachers, and the industry expect certain knowledge and skills that are related to abilities such as learning-to-learn, thinking, adaptability, problem solving, positive attitude etc. These competencies would cover both cognitive and affective domains.

A precedence diagram for the subjects is drawn where the prerequisites for each subject are graphically illustrated. The number of levels in this diagram is determined by the duration of the course in terms of number of semesters etc. Using the precedence diagram and the time duration for each subject, the curriculum is organized.

- Design & development of instructional materials

The content outlines are developed by including additional topics that are required for the completion of the domain and for the logical development of the competencies identified. Evaluation strategy and scheme is developed for the subject. The topics are arranged/organized in a meaningful sequence.

The detailed instructional material – Training aids, Learner material, reference material, project guidelines, etc.- are then developed. Rigorous quality checks are conducted at every stage.

- Strategies for delivery of instruction

Careful consideration is given for the integral development of abilities like thinking, problem solving, learning-to-learn etc. by selecting appropriate instructional strategies (training methodology), instructional activities and instructional materials.

The area of IT is fast changing and nebulous. Hence, considerable flexibility is provided in the instructional process by specially including creative activities with group interaction between the students and the trainer. The positive aspects of Web based learning –acquiring information, organizing information and acting on the basis of insufficient information are some of the aspects, which are incorporated, in the instructional process.

- Assessment of learning

The learning is assessed through different modes – tests, assignments & projects. The assessment system is designed to evaluate the level of knowledge & skills as defined by the learning objectives.

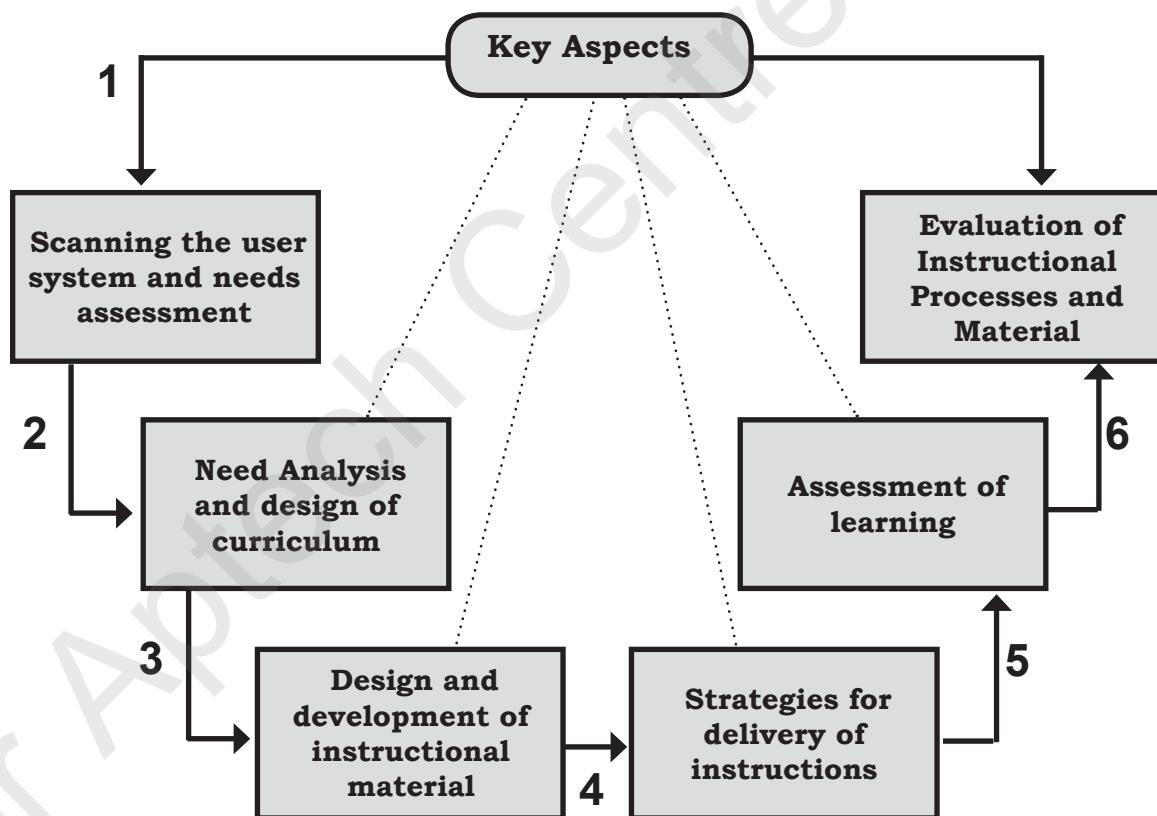
- Evaluation of instructional process and instructional materials

The instructional process is backed by an elaborate monitoring system to evaluate - on-time delivery, understanding of a subject module, ability of the instructor to impart learning. As an integral part of this process, we request you to kindly send us your feedback in the reply pre-paid form appended at the end of each module.

*TAG – Technology & Academics Group comprises of members from Aptech Ltd., professors from reputed Academic Institutions, Senior Managers from Industry, Technical gurus from Software Majors & representatives from regulatory organizations/forums.

Technology heads of Aptech Ltd. meet on a monthly basis to share and evaluate the technology trends. The group interfaces with the representatives of the TAG thrice a year to review and validate the technology and academic directions and endeavors of Aptech Ltd.

Aptech New Products Design Model



“

A little learning is a dangerous thing,
but a lot of ignorance is just as bad.

”

Preface

The book 'Web Component Development Using Java' Trainer's Guide aims to teach the students how to develop Web applications using Java Web components such as Servlets and JavaServer Pages (JSP). The faculty/trainer should teach the concepts in the theory class using the slides. This Trainer's Guide will provide guidance on the flow of the module and also provide tips and additional examples wherever necessary. The trainer can ask questions to make the session interactive and also to test the understanding of the students.

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 Certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

We will be glad to receive your suggestions.

Design Team

“ Practice is the best of
all instructors.



Table of Contents

Sessions

1. Introduction to Web Applications
2. Java Servlet
3. Session Tracking
4. Filters and Annotations
5. Database Access and Event Handling
6. Asynchronous Servlet Communication
7. JavaServer Pages
8. JSP Implicit Objects
9. Standard Actions and JavaBeans
10. Model-View-Controller Architecture
11. JSP Expression Language
12. JavaServer Pages Standard Tag Library
13. JSP Custom Tags
14. Internationalization
15. Securing Web Applications



“ The future depends on what
we do in the present.

Session 1 – Introduction to Web Applications

1.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. Prepare a question or two which will be a key point to relate the current session objectives.

1.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the purpose of different applications
- Explain Web applications and their advantages
- Describe the architecture and components of Web application
- Describe the role of HTTP protocol and its methods used for accessing Web pages
- Explain the use of Common Gateway Interface (CGI) language
- Explain the different types of components used in developing Web application
- Explain the advantages and disadvantages of Servlets
- Explain the services provided by a Web container
- Describe the life cycle and directory structure of a Web application
- Explain how to package a Web application
- Develop a Web application in NetBeans Integrated Development Environment (IDE)

1.1.2 Teaching Skills

To teach this session successfully, you should be aware with the working and the concepts of Internet, World Wide Web (WWW), and its related technologies such as HyperText Markup Language (HTML) protocol. You should be familiar with Web applications, its architecture, and request-response cycle. Also, familiarize yourself with different types of Java Web components involved in developing Web applications. Apart from this, you should acquaint yourself with the the working of other server-side technologies such as Common Gateway Interface (CGI), Active Server Pages (ASP), and so on.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives
<ul style="list-style-type: none">❖ Explain the purpose of different applications❖ Explain Web applications and their advantages❖ Describe the architecture and components of Web application❖ Describe the role of HTTP protocol and its methods used for accessing Web pages❖ Explain the use of Common Gateway Interface (CGI) language❖ Explain the different types of components used in developing Web application❖ Explain the advantages and disadvantages of Servlets❖ Explain the services provided by a Web container❖ Describe the life cycle and directory structure of a Web application❖ Explain how to package Web application❖ Develop a Web application in NetBeans Integrated Development Environment (IDE)

Tell them that they will be introduced to the basics concept of Web applications. Tell them that they will be explained about the architecture of Web application, request-response mechanism offered by HTTP protocol, Web application life cycle, and the role of Java in Web application developments, and its component. The session also covers the directory structure of Web applications.

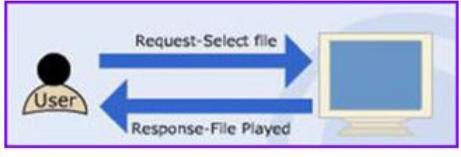
The session also explains server-side technologies such as CGI, ASP, Java Servlets, JSP, and so on and their role in developing dynamic Web applications. Finally, it discusses about packaging of Web application in the Web archive file and building and testing of Java Web application in NetBeans IDE.

1.2 In-Class Explanations

Slides 3 and 4

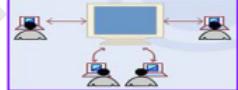
Let us understand different types of applications.

Introduction 1-2

- ❖ An application is a collection of programs designed to perform a particular task.
- ❖ **Desktop Application**
 - Runs only in a local machine.
 - Can neither be viewed nor be controlled by other users.
 - Example: Microsoft Word.
 - Figure depicts the desktop application.

© Aptech Ltd. Web Component Development Using Java/Session 1 3

Introduction 2-2

- ❖ **Networking Application**
 - Runs in a Local Area Network (LAN), Metropolitan Area Network (MAN), or World Area Network (WAN).
 - Can be viewed and controlled by users in that particular network only.
 - Example: Novell Netware.
 - Figure depicts the networking application.
- ❖ **Web Application**
 - Runs at a remote location.
 - Can be viewed and controlled by all the users having administrative or equivalent privilege, throughout the globe, at any instance of time.
 - Example: Internet mail services, such as www.yahoo.com.
 - Figure depicts the Web application.

© Aptech Ltd. Web Component Development Using Java/Session 1 4

Use slides 3 and 4 to introduce and explain different types of applications.

Ask the students what they know about applications and ask them to give few examples of applications they have seen or used. After listening to their answers, tell the students that an application is a collection of programs designed to perform a particular task. Application software can run only with the aid of a system software. Microsoft Word, Excel, PowerPoint, Notepad, Web browsers, and so on are some of the examples of application software. Different types of applications are designed for different purposes. For example, for accounting purposes, Tally is one of the most popular applications used in the market.

Based on the environment in which an application is run and their accessibility by the users, they are classified as desktop application, networking application, and Web application.

Use the figure in slide 3 to display and explain desktop application. Some examples of desktop applications are: Microsoft Paint, Calculator, Games, Multimedia Player, and so on.

Use slide 4 to explain about the network applications. A Network application is any application running on one host and provides a communication to another application running on a different host, the application may use an existing application layer protocols such as HTTP, Simple Mail Transfer Protocol (SMTP), and so on. Few examples of network applications are: Email clients, File Transfer Protocol (FTP), Instant Messaging, Internet Telephony, and so on. Explain the network applications with the help of the figure shown on slide 4.

Then, explain about Web applications. Web applications are the type of network applications accessed on World Wide Web. A Web application runs on a Web browser. Some of the popular examples of Web applications are: online shopping sites, online auction sites, social sites such as facebook, linkedin, and so on. Explain the figure showing the storage of Web applications on Web server and its access by clients or users of that application.

Slides 5 and 6

Let us understand Web applications in detail.

Web Applications 1-2

- ❖ A Web application:
 - Is a software application that runs on a Web server.
 - Consists of Web pages which can be static and dynamic.
- ❖ Static Web pages:
 - Are created using Web technologies such as HyperText Markup Language (HTML), Cascading Style Sheet (CSS), and JavaScript.
- ❖ Dynamic Web pages:
 - Server-side programs are used to achieve dynamic functionalities on Web pages.
 - Provide interactivity with the users.

Users accessing the Web pages on the World Wide Web (WWW) are referred to as Web clients.

© Aptech Ltd. Web Component Development Using Java/Session 1

Web Applications 2-2

- ❖ The advantages of Web applications over desktop applications are as follows:
 - Easier access to information
 - Lower maintenance and deployment costs
 - Platform independence
 - Wider visibility

© Aptech Ltd. Web Component Development Using Java/Session 1

Use slides 5 and 6 to explain Web applications. Tell them that a Web application is a software application that runs on a Web server.

Here, you can ask the students, what they understand by a Web site? After listening to their answers, tell them that a Web site that a user visits is actually a Web application running in a Web server at a remote location and executed in a Web browser, for example, www.wikipedia.com.

Tell them that a Web application consists of Web pages which can be static and dynamic. Static Web pages are viewed by the user exactly as stored in the server and are created using Web technologies such as HyperText Markup Language (HTML), Cascading Style Sheet (CSS), and JavaScript. However, to achieve dynamic functionalities on Web pages, server-side programs are used. The dynamic Web pages provide interactivity with the users. For example, an online bookstore Web application allows user to select and deselect books on the Web page.

Use slide 6 to describe the advantages of Web applications over desktop applications. Tell them that Web applications have the following advantages over desktop applications:

- **Easier access to information** - The protocol used for accessing a Web application is Hypertext Transfer Protocol (HTTP) and is supported by most of the Operating System (OS). Moreover, the client software required is just a Web browser, which now comes packaged with most of the OS. Desktop applications are kept in a specific location and cannot be accessed by everyone. Web applications can be accessed by anyone from any location using the Internet.
- **Lower maintenance and deployment costs** - Unlike desktop applications, Web applications run in a Web browser and do not need client software to be installed on every client system. Web application code can be modified and maintained at the server end. This saves time and cost involved in upgrading and deploying the applications.
- **Platform independence** - Web applications run in Web browsers and therefore, can be accessed on any machine with different OS.
- **Wider visibility** - The Web application can easily be accessed around the globe at any instance of time.

Slides 7 and 8

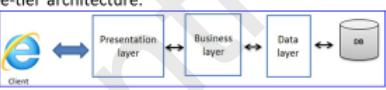
Let us understand Web application architecture.

Web Application Architecture 1-2

- ❖ **One-tier architecture**
 - Code related to presentation, business, and data access logic are all clubbed together.
 - Figure shows the one-tier architecture.
- ❖ **Two-tier architecture**
 - Code related to data access logic is separated from the other two components.
 - Any interaction with data tier will be done through business tier.
 - Figure shows the two-tier architecture.

© Aptech Ltd. Web Component Development Using Java/Session 1 7

Web Application Architecture 2-2

- ❖ **Three-tier architecture**
 - Code related to all three components is separated from each other.
 - Business tier acts as an interface between the data tier and the presentation tier.
 - Figure shows the three-tier architecture.
- ❖ **N-tier architecture**
 - Layers are further subdivided for ease of functioning.
 - Presentation layer is a graphical user interface that displays data to users.
 - Business layer and application layer are separated reducing the number of locations implementing the logic.
 - Figure shows the N-tier architecture.

© Aptech Ltd. Web Component Development Using Java/Session 1 8

Use slides 7 and 8 to explain Web application architecture.

Tell the students that a Web application basically has three components: the presentation layer, the application layer, also known as business layer, and the data access layer. The architecture of an application defines how these three components will be clubbed together and their interaction with each other. Explain them that layering a Web application logically into presentation, business, and data access layers helps you to create maintainable code and allows you to monitor and optimize the performance of each layer separately. A clear logical separation also offers more choices for scaling your application.

Use the first figure on slide 7 to explain one-tier architecture. Tell them that in one-tier architecture, the code related to presentation, business, and data access logic are all clubbed together. Then, use the second figure on slide 7 to explain two-tier architecture. Tell them that in two-tier architecture, the code related to data access logic is separated from the other two components in the data access layer.

Then, use the first figure on slide 8 to explain three-tier architecture. Tell them that in three-tier architecture, code related to all three components is separated from each other. However, the business tier acts as an interface between the data tier and the presentation tier. Normally, Web applications are developed with three-tier architecture.

Finally, if the number of layers or tiers are increased based on simplifying the functionality of the Web application architecture, it is referred to as N-tier architecture. Use the second figure on slide 8 to explain N-tier architecture.

In-Class Question:

After you finish explaining Web application architecture, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



In Web application architecture, which layer is also known as the business layer?

Answer:

Application layer.

Tips:

Traditional applications consist only of one-tier, which resides on the client machine, but web applications adopt n-tiered approach. As a result, many variations were possible. The most common structure is the three-tiered application architecture.

However, for more complex applications, a 3-tier solution may fall short, and it may be beneficial to use an n-tiered approach, where the greatest benefit is breaking the business logic, which resides on the application tier, into a more fine-grained model. Another benefit may be adding an integration tier that separates the data tier from the rest of tiers by providing an easy-to-use interface to access the data.

Slide 9

Let us understand the language of Web application technology, HTML.

Web Application Technology - HTML

- ❖ HTML is a presentation language which enables Web designer to create visually attractive Web pages. These pages are stored on Web.
- ❖ HTML allows Web designers to add images, create forms, and embed media objects on the Web pages. All these are referred as Web resources which are stored on the Web server along with HTML pages.
- ❖ Figure shows the Web page designed using HTML.

```
<html>
  <body>
    <h1> Sample HTML Page </h1>
    <!-- Displaying an image -->
    </p>
    <p> This Web page is created in <b> HTML </b> language which has following features: </p>
    <!-- Displaying a list -->
    <ol>
      <li> Stands for HyperText Markup Language </li>
      <li> A markup language </li>
      <li> Contain HTML tags and plain text </li>
      <li> Also called as web pages </li>
    </ol>
  </body>
</html>
```

© Aptech Ltd. Web Component Development Using Java/Session 1

9

Use slide 9 to explain HTML Web application technology.

Ask them what the expansion of HTML and HTTP?.

After listening to their answers, tell them that HTML stands for HyperText Markup Language and HTTP stands for Hypertext Transfer Protocol.

HTML is the standard markup language used to create web pages. HTML is a presentation language which enables Web designers to create visually attractive Web pages. These pages are stored on Web server. The users accessing the Web pages on the WWW are referred to as Web clients.

HTML allows Web designers to add images, create forms, and embed media objects on the Web pages. All these are referred as Web resources which are stored on the Web server along with HTML pages.

Then, use the figure on slide 9 to explain the Web page designed using HTML. Tell them that HTML is written in the form of HTML elements consisting of tags enclosed in angle brackets. For example, `<h1>` and `</h1>`. A Web browser can read HTML tags and interprets them into visible formatted content displayed on the Web page.

Tips:

A markup language is a computer language that describes how a Web page should be formatted. It supports tags which are simple instructions that tell the Web browser how a page should look when it is displayed by the browser.

Then, tell them that the Web pages are transferred between the Web server and clients through HTTP protocol.

Slide 10

Let us understand the HTTP protocol.

Web Application Technology - HTTP

- ❖ The requests and responses sent to a Web application from one computer to another computer are sent using HTTP.
- ❖ An HTTP client, such as a Web browser opens a connection and sends a request message to an HTTP server asking for a resource.
- ❖ The server, in turn, returns a response message with the requested resource. Once the resource is delivered, the server closes the connection.
- ❖ Referred to as a stateless protocol as HTTP does not store any connection information about the page.
- ❖ Figure depicts HTTP request and response to a Web application.

© Aptech Ltd. Web Component Development Using Java/Session 1 10

Use slide 10 to explain HTTP protocol used in transferring request and response in Web applications. The main protocol behind the WWW is the HTTP forms the basis for the transfer of documents between Web server and client. It runs top of the TCP/IP protocol suite.

Tell them that HTTP defines how messages are structured and conveyed and what responses Web servers and browsers should give to different requests. For example, when you enter a Uniform Resource Locator (URL) address in a Web browser, a HTTP request message is sent to the Web server asking it to process the data sent from the client and communicate back with the generated response from the server. Thus, HTTP functions as a request-response protocol in the client-server computing model.

Use the figure on slide 10 to explain HTTP request and response to a Web application.

Tips:

1. URL

A URL is a Web address that is used with HTTP protocol to access the resources from the Web server. A typical URL might look like `http://www.aptech-education.com/`.

2. The HTTP Protocol

At the time when the WWW started gaining popularity, there were a lot specialized data transfer protocols related to file transfers, news broadcast, mail transfer, search and retrieval facilities. However, the WWW has its own specific needs such as file transfer functionality, search facilities with indexing options, automatic format negotiation and the ability to refer the client to another server. This initiated the development of HTTP.

The Web server is an ‘HTTP’ server. Any HTTP server responds to the browser on request. HTTP is used in the application area of networking. It is a high level rare, which defines how the Web browser communicates with the Web server.

HTTP is a ‘Connectionless’ protocol. By connectionless we mean that the connection is closed as soon as the client’s requirement is over. By working with connectionless protocol, the client opens the connection, sends its request to the server, receives a response and then closes the connection. So each request has to open a new connection. On the contrary, a ‘connection-oriented’ protocol holds the connection, even after responding in order to service future request. For example, FTP is a connection-oriented protocol.

HTTP is a ‘stateless’ protocol. A ‘stateless’ protocol do not retain the information about prior connections, and the protocol request does not distinguishes one clients request from the another. In HTTP connection, a new sessions opens that sends its request to the server. In HTTP every client connection opens a new session that sends its request to the server. So the server receives the request from an anonymous client every time. This stateless nature keeps the protocol very simple and straight forward. This consumes very few resources on the server and can support more simultaneous users since there are no client information overheads to be maintained throughout the sessions. In contrast, a protocol sharing state such as FTP maintains the overheads of keeping users information throughout the sessions.

Slides 11 and 12

Let us understand HTTP request.

HTTP Request 1-2

- ❖ The request message sent by the client is a stream of text.
- ❖ Figure depicts the request message structure.

❖ Its elements are namely, Request line and Header information.

❖ **Request line**

- Contains the HTTP method, the address of the Web page or document referred to as Uniform Resource Locator (URL), and the HTTP version.
- Figure shows the request line of the request message.

HTTP Method	HTTP Version
Request Line	HTTP/1.1
GET/ index.html	
Address of document	

© Aptech Ltd. Web Component Development Using Java/Session 1 11

HTTP Request 2-2

- ❖ **Header information**

- Specifies the User-Agent along with the Accept header.
- The User-Agent element header indicates the browser used by the client.
- The Accept header element provides information on what media types the client can accept.
- After the header, the client sends a blank line indicating the end of request message.
- Figure shows the sample header information sent in the request message.

```
User-Agent: Mozilla/4.0 (compatible; MSIE 4.0; Windows 95)
Accept: image/gif, image/jpeg, text/*, */*
```

© Aptech Ltd. Web Component Development Using Java/Session 1 12

Use slides 11 and 12 to explain HTTP request.

Use the first figure on slide 11 to explain the request message structure. Use the second figure in slide 11 to display and explain the request line of the request message.

The beginning of the HTTP request will have the request line, which will be followed by up to three headers: a general header, a request header, and an entity header. After that will be the message body. The request line specifies the method type (such as GET or POST), the URI requested, and the version of HTTP to use.

The different methods types can be as follows:

- OPTIONS
- GET
- HEAD
- POST
- PUT

- DELETE
- TRACE

The general header has basic information about the connection, such as cache details, the date, and how the data are encoded during transfer. The request header is specific to what a request needs; it defines things such as what languages, encodings, and character sets the client is willing to accept.

The request header also specifies the hostname being accessed which is very important for virtual Web servers, the range (for requests that are getting partial bytes for things like download resumption), the referrer (what URL lead you here), and the user agent. There are other fields in the request header as well. The message headers are a set of name/value pairs.

The entity headers provide information about the entity body (which is really just the message body after being decoded) that the request will be providing (such as POST data or a file that is being uploaded). This data includes the content length in bytes and the content type. If there is no entity following the headers (like in a GET request), these entity headers are not needed.

Then, use the figure on slide 12 to display a sample header information sent in the request message.

Slides 13 and 14

Let us understand HTTP Response.

HTTP Response 1-2

- ❖ The Web application processes the request sent by a client and generates a response message for the requesting client.
- ❖ Figure depicts the response message structure.

Its elements are namely, Status line and Header information.

- ❖ **Status line**
 - Indicates status of the requested process.

© Aptech Ltd. Web Component Development Using Java/Session 1 13

HTTP Response 2-2

- ❖ **Header information**
 - Contains information such as server, last modified date, content-length, and content-type.
 - Figure shows the sample message format sent in the response with the HTML content from the Web server.

```

HTTP/1.1 200 OK
Server: APACHE
Content-Type: text/html
Content-Length: 370
Last-Modified: Mon, 17 Jun 2014 10:14:49 GMT
    ← HTTP Response

<html>
<body>
    <h1> Sample HTML Page </h1>
    <!-- Displaying an image -->
    An image:
    </p>
    <p>
        This Web page is created in <b> HTML </b> language which has following
        features:
    </p>
    <!-- Displaying a list -->
    <ul>
        <li> HTML stands for Hyper Text Markup Language </li>
        <li> It is a markup language </li>
        <li> Contains HTML tags and plain text </li>
        <li> Also called as web pages </li>
    </ul>
</body>
</html>

```

© Aptech Ltd. Web Component Development Using Java/Session 1 14

Use slides 13 and 14 to explain HTTP response. Use the figure in slide 13 to display response message structure.

Tell them just as the HTTP request started with a request line to indicate the most basic information, the HTTP response begins with a status line.

The status line informs the consumer of the HTTP version in use, a status code (such as 404 or 200), and a reason explaining the code. After the status line are the general headers, the response header, and the entity headers, followed by the message body.

The response headers contain information specific to the needs of the response, such as the location (used to redirect the client to a new URI), content-type, date, content-length, and so on.

Message Body

The message body part is optional for an HTTP message. However, if it is available then it is used to carry the entity-body associated with the request or response. If entity body is associated then usually **Content-Type** and **Content-Length** headers lines specify the nature of the body associated.

A message body is the one which carries actual HTTP request data that includes form data and HTTP response data from the server that includes content, files, images, and so on.

Then, explain the figure shown on slide 14 which is generated to be sent in the response to the client.

Tips:

The status line includes a numeric status code, such as "200" and reason message. The official registry of HTTP status codes is maintained by the Internet Assigned Numbers Authority (IANA). The first digit of the Status-Code has five values and defines the class of response. The last two digits do not have any categorization role. The five values of the first digit are:

- 1xx: Informational - Request received, continuing process
- 2xx: Success - The action was successfully received, understood, and accepted
- 3xx: Redirection - Further action must be taken in order to complete the request
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfil an apparently valid request

Slides 15 to 18

Let us understand HTTP methods.

HTTP Methods 1-4

- ❖ Web applications allow users to enter information using forms and send to the server for processing.
- ❖ The data entered into the fields in a form is sent to the Web server through URL.
- ❖ The data is processed on the Web server and the appropriate response is generated which is sent back to the user.
- ❖ Figure shows an example of request parameters.



© Aptech Ltd.

Web Component Development Using Java/Session 1

15

HTTP Methods 2-4

The HTTP request messages commonly use the following HTTP methods for transmitting request data over the Web:

- ❖ **GET**
 - Informs the server to retrieve information from the request URL.
 - Information is passed as a sequence of characters that are appended at the end of the request URL, which forms a query string.
 - The length of query string is restricted to 240 to 255 characters depending on the server.
 - A sample query string constructed while sending user data in the request URL:
`http://www.abcBank.co.uk/search?name=john&pass=j7652`
- ❖ **HEAD**
 - Functionally same as GET, except that the client uses HEAD method to receive only the headers of the response and not the message body.
 - Advantageous when the user want to check characteristics of a resource without actually downloading the complete resource.

© Aptech Ltd.

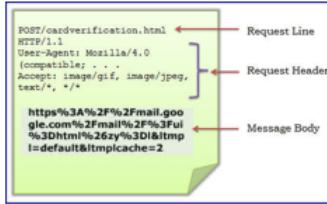
Web Component Development Using Java/Session 1

16

HTTP Methods 3-4

❖ POST

- Used when sending information, such as credit card numbers or information to be saved in the database.
- Data sent is in encrypted format and not visible to the client, and there is no limit on the amount of data being sent.
- The data is passed to the server in the body of the HTTP message, and hence the request cannot be bookmarked or emailed.
- Figure shows the request structure with HTTP POST method.



© Aptech Ltd.

Web Component Development Using Java/Session 1

17

HTTP Methods 4-4

❖ **PUT**

- ❑ Used to request the server to store the data enclosed in the HTTP message body at a location provided in the request URL.

❖ **DELETE**

- ❑ Used to request the server to delete file at a location specified in the request URL.

❖ **OPTIONS**

- ❑ Can be used to query the server on the methods supported for a particular resource available on the server.

❖ **TRACE**

- ❑ Basically used for debugging and testing the request sent to the server.
- ❑ It asks the user to echo back the sent request.
- ❑ Useful when the request sent to the server reaches through the proxies.

© Aptech Ltd.

Web Component Development Using Java/Session 1

18

Use slides 15 to 18 to explain HTTP methods. Give them an example of an online banking site, in which the user enters username and password. The information is sent to the Web server through the request parameters and after validating the data received with the database, the account information is sent as a response. The server treats the data entered in each field as a request parameter. The server can extract the values of each request parameter in the request for processing.

In slide 16, tell them that the HTTP methods specified in the `request-line` element of the request indicate the type of operation to be performed on the resource.

Provide an example for the `GET` method. Tell them in search engine such as `www.google.com` uses the `GET` method to retrieve search results for search strings entered by user.

In slide 16, explain the sample query string constructed while sending user data in the request URL, `http://www.abcBank.co.uk/search?name=john&pass=j7652`. Tell them that the length of query string is restricted to 240 to 255 characters depending on the server. Thus, this method cannot be used to send large amount of data from the forms. Also, the data sent on the URL is visible; hence, this method is not reliable to send the important information such as credit card number or password.

Then, explain them the use of the `POST` method. Take an example of transferring money from one bank account to another which needs secure transaction. It should not repeat the process without user approval. Thus, if you try to refresh the page during transaction, if the method used in the Web application is `POST`, then a display message is generated that warns you that there may be side effects, as the transaction is not successful.

The `POST` request message has a content body that is normally used to send parameters and data. Unlike using the request URI, there is no upper limit on the amount of data that can be sent and `POST` must be used if files or other variable length data has to be sent to the server. Use the figure on slide 17 to explain the request structure with HTTP POST method.

Apart from HTTP request types GET and POST, there are other HTTP request types such as HEAD, PUT, TRACE, and DELETE. A HEAD request returns HTTP headers only, PUT and DELETE allow clients to create and remove resources from the Web server, and TRACE returns the request headers to the client.

In-Class Question:

After you finish explaining HTTP methods, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which HTTP method is used to retrieve information from the request URL?

Answer:

GET method.

Slide 19

Let us understand Java Web application.

Java Web Application

- ❖ The Web components of Java Web application are as follows:
 - **Servlet:** Java classes that dynamically process HTTP requests and construct responses, a key component in Web development.
 - **JSP:** A text document that creates dynamic Web content, usually to display the content in the Web browser.

- ❖ Prevalent Server-side Technologies:
 - Common Gateway Interface (CGI)
 - Proprietary Web server API (ISAPI)
 - Server-side JavaScript
 - Active Server Pages
 - PHP
 - Java Servlets
 - Java Server Pages (JSP)

© Aptech Ltd. Web Component Development Using Java/Session 1 19

Use slide 19 to explain Java Web application. Tell the students that the potential of Java as a server-side development platform extends the power of the Web in creating Web applications. However, the first release of Java did not support the APIs for developing Web application, but was intended for use in embedded devices. Also, during that time, even the Web pages contained largely static content. Soon with the development of Java Applets, which support the execution of Java code within the browsers, Java came out as a powerful platform for developing Web components accessed on multiple platforms providing dynamic functionality to the Web pages. Web components are functional software running on the Web server which handles the incoming HTTP request and provides responses.

After defining the Web components of Java Web application, tell them that the Java Web technology named Servlet formed a key component in Web development. However, before Servlets, there were technologies such as Common Gateway Interface (CGI), Active Server Pages (ASP), and so on which were practiced for generating dynamic content for the pages.

List to the students the various prevalent server-side technologies listed on the slide. Some of them are as follows:

Microsoft and Netscape APIs

Microsoft and Netscape, each developed their own APIs that allows developers to write server applications very quickly. The server application can be used as shared libraries. Several libraries are loaded into the same process area with the Web server. These libraries are able to service multiple requests without creating a new process. These can either be loaded when they are needed. Once any of these have been idle for a set amount of time, they are unloaded by the Web server from the memory.

The server-side uses JavaScript as its scripting language. It is an extension of the core JavaScript language with features such as database access, session management, inter-operability with server-side Java classes, using Netscape LiveWare technology. The compiled server-side JavaScript is not platform specific, however, specific to Netscape's HTTP servers.

Personal Home Page (PHP)

Like JavaScript, PHP provides strong support for pattern matching and database access. It provides extension for communicating with other networks resources such as mail and directory servers.

PHP is an open source technology. It is compatible with Windows NT as well as UNIX operating system. PHP applications run on a number of HTTP servers namely, Apache, Microsoft IIS, and Nestcape Enterprise Server.

Active Server Pages (ASP)

ASP supports multiple scripting languages, such as Jscript and VBScript. Jscript is based on JavaScript. VBScript is based on Microsoft Visual Basic programming language. VBScript includes support for accessing ActiveX components which are compiled code objects that can encapsulate virtually any functionality including database access and file manipulation. The major drawback of ASP is that it can run only on the Internet Information Server (IIS) under the Windows platform.

Slides 20 and 21

Let us understand Common Gateway Interface (CGI).

Common Gateway Interface (CGI) 1-2

- ❖ A set of standards that specify how data is transferred between the Web server and server-side CGI programs.
- ❖ The data provided by the client in the Web page is sent from HTTP server to the gateway or CGI program.
- ❖ The CGI program processes the data and returns the result to the Web server, which in turn sends it to the client.
- ❖ Figure depicts the server process for running CGI.

A program that gives dynamic output, thereby executing in real time, is written at the server-side using standards of CGI.

© Aptech Ltd. Web Component Development Using Java/Session 1 20

Common Gateway Interface (CGI) 2-2

- ❖ When a Web server receives a request that need to access a CGI program, it creates a new instance or process of the CGI program and then pass the client data to it.
- ❖ Figure depicts the working of CGI program.

- ❖ Disadvantages of CGI:
 - ❑ Reduced efficiency - Number of processes that can be executed by a Web server is limited to each server.
 - ❑ Reloading Perl interpreter - Each time the server receives a request, the Perl interpreter needs to be reloaded. This reduces the efficiency of the server.

© Aptech Ltd. Web Component Development Using Java/Session 1 21

Use slides 20 and 21 to explain Common Gateway Interface (CGI). Tell them that CGI is one of the most popular server-side technologies used to create dynamic Web content. Web server implementing CGI acts as a gateway between the user request and the data it requires. These programs are known as CGI scripts or simply CGIs, and are typically written in a scripting language. They can also be written in any programming language. CGI programs and scripts are usually collected in a folder named /cgi-bin/.

Use the figure on slide 20 to display and explain the server process for running CGI. Tell them that when a Web server receives a request that need to access a CGI program, it creates a new instance or process of the CGI program and then pass the client data to it. This means that for every new request, the server creates a separate instance of CGI program which is responsible to generate a response for its request.

Use the figure on slide 21 to explain the working of CGI program. Tell them that creating a process for each request occupies server resources, thus, limiting the number of requests a server can handle concurrently.

Use slide 21 to explain the disadvantages of CGI. Tell them that CGI as a server-side scripting language, has certain disadvantages. After explaining reduced efficiency, tell them that each time a process is executed, different instances of the same processes is created on the server, thereby, resulting in overloading of server and reducing server efficiency.. Before describing the next disadvantage, tell them that the widely accepted platform for writing CGI script is Perl. Perl is a feature-rich, dynamic, and high-level programming language. The languages in the Perl family include Perl 5 and Perl 6.

In-Class Question:

After you finish explaining CGI, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which is the widely accepted platform for writing CGI script?

Answer:

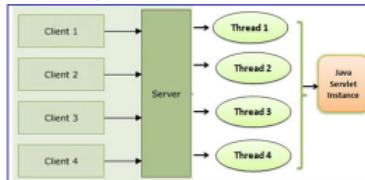
Perl.

Slide 22

Let us understand Servlets.

Servlets

- ❖ The processing of Servlet is very similar to CGI program, as it also responds to HTTP request and generates dynamic response.
- ❖ Servlet:
 - ❑ There is only a single instance of Servlet created on the Web server.
 - ❑ To service multiple clients' request, the Web server creates multiple threads for the same Servlet instance.
 - ❑ Each thread handles request received from the client and generates response that is sent to the Web engine, which in turn sends the response to the client.
- ❖ Figure shows the working of the Servlet.



© Aptech Ltd. Web Component Development Using Java/Session 1 22

Use slide 22 to explain Servlets.

In 1996 Sun introduced Servlets as small Java-based applications for adding dynamic functionalities to Web server. Java Servlets have a programming model similar to CGI scripts.

Unlike traditional CGI programs that require spawning a new process to handle each new request, the Servlets spawn new threads for new requests and these threads run inside a single process on the Web server. This process runs on a Java Virtual Machine (JVM) which is a platform specific program for running compiled Java programs on the Web server. Instead of creating a new process for each request, the JVM creates a Java thread to handle each Servlet request.

Tell them that Java threads are lightweight and they execute within the processor memory already allocated by the JVM. This makes Servlet execution considerably more efficient than CGI processing.

Use the figure on the slide to explain the working of a Servlet. Tell them that the initialization code in Servlets is executed only once at the beginning of the program and is refreshed automatically on receiving separate requests. Whereas in CGI, separate instances of the program are being invoked each time a process is executed. The multi-threading property of servlets helps in handling separate requests by allocating resource to separate threads. This gives a boost to the performance, thereby, increasing the efficiency of servlets.

Then explain them that Servlets provide a Java-based Servlet API for mapping HTTP requests and responses on the server. Servlet generates dynamic Web content using Java code that outputs HTML or other data for sending responses to the clients.

Slide 23

Let us understand the merits of Servlet.

Merits of Servlet

- ❖ Servlets are written using Java language, which makes extensive use of Java API. The merits of servlets are:
 - ❑ **Enhanced efficiency** - Servlet is required to be executed only once at the beginning with the initialization code. Thereafter, it gets auto refreshed each time a request is sent for execution.
 - ❑ **Ease of use** - Servlets does not have too many complexities. It is just basic Java along with HTML implemented within the Java code, which increases the ease of use.
 - ❑ **Powerful** - The usage of Java APIs makes the Servlets a powerful server-side scripting language.
 - ❑ **Portable** - Java is platform independent and since Servlet uses Java code for writing scripts, it can be executed in any platform.
 - ❑ **Safe and cheap** - Usage of Java codes provides high security of data to be sent and received, thereby maintaining the safety.

© Aptech Ltd. Web Component Development Using Java/Session 1 23

Use slide 23 to explain the advantages of Servlet.

The Web server loads and executes Servlet in the same manner, as the browser does with an Applet. However, unlike Applets, Servlets are not responsible to display GUI to the user. The Servlet works on the server side and sends the processing result to the client. The processing result is usually in the form of an HTML file. Servlets provide a framework for creating server-side codes that work with the request and response architecture of the Web server.

Some of the advantages of Servlet technology are as follows:

Efficiency – Servlets are loaded in the same process as the Web server. This is an advantage, since it overcomes the overheads of creating new process for every request. The multithreading properties of Servlet ensures that several threads are produced within the same server process. These threads can service all the clients request.

Ease of Use – Servlets can be executed more quickly than common scripting languages, as Servlets are compiled into Java byte code. Byte-code compilation improves the performance of Servlets through compiled time code optimizer. Moreover, compilation offers the advantages of strong error and type checking. Lastly, compiled code is more secured than non-complied code.

Powerful – Servlets are almost crash proof. This is because Servlets are written in Java and Java Virtual Machine (JVM) does not allow the Servlets to access the memory directly. Therefore, the possibility of invalid memory access is reduced. Even a poorly designed Servlet cannot crash the system because the JVM checks the validity of all compiled Java class files and does not allow any illegal operation to be performed. Moreover, instead of crashing, the JVM will raise an exception. If this exception is not caught, then the JVM will itself handle the exception.

Portable – Servlets are written in Java. The Servlet APIs are widely accepted. They are highly portable across the platform. So, a Servlet is written on Windows NT platform running Java Web server can easily be deployed on a UNIX machine running Apache Web server.

Safe and Cheap – Servlets are Secure. There are important features that ensure Servlet security. Firstly, Servlets use the server security manager, since the security manager can restrict network or file access for untrusted Servlets. The security manager can give full access or digital signed and trusted Servlets. Secondly, Servlets are secured because they can only be invoked through the Web server and the Web server is protected behind the firewall security.

Tips:

Servlets are durable objects. Therefore, they remain in the memory until they are destroyed explicitly. So, Servlets need to instantiated once at the beginning and the same durable object can be shared across many users. For example, a single database connection can be created using a Servlet when it is first loaded, and the same can be shared across all requests.

Servlets are completely protocol independent. Although, they are commonly used on HTTP server. They can also support FTP, SMTP, POP3, and the other protocols.

Slides 24 and 25

Let us understand developing Web applications.

Developing Web Applications 1-2

- ❖ Java Web application comprises servlets, JSP pages, images, HTML files, JavaBeans, Applets, and Java classes.
- ❖ Package all the files associated with Web application into a single Web archive file (war).
- ❖ The Java Web applications are deployed on a Java-enabled Web server such as Tomcat.



Apache Tomcat

- ❖ Examples of other Java servers containing a Web container for managing Web components include:



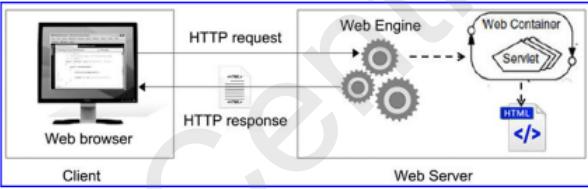





© Aptech Ltd. Web Component Development Using Java/Session 1 24

Developing Web Applications 2-2

- ❖ Figure shows a Web server containing a Web container used for processing the HTTP request and HTTP response for the accessed Servlet or JSP.



Client Web Server

- ❖ Every Java Web server contains a Web container which is responsible for managing the Web components on the server.

© Aptech Ltd. Web Component Development Using Java/Session 1 25

Use slides 24 and 25 to explain developing Java Web applications. Tell them that an application registered into World Wide Web (WWW) and accessible to its users from a public server is a Web application or a Web app. It is a software that runs in a Web browser and is developed in a browser-supported programming language and depends on a Web browser to render the application. Java Web application comprises servlets, JSP pages, figures, HTML files, JavaBeans, Applets, and Java classes. If a Web application is to be ported to some other server or system, a developer has to copy or move all these files to the new system. However, an easy alternative is to package all the files associated with Web application into a single Web archive file (war) and deploy this war file on the Web server.

There are many vendors which provide the Java Web server with Servlet and JSP specification implemented on them. Some of the vendors are namely, Tomcat, GlassFish, and JBoss.

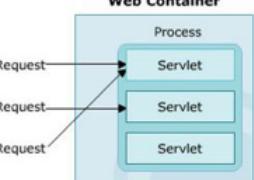
Use the figure in slide 25 to explain that every Java Web server contains a Web container. The Web container is used for processing the HTTP request and HTTP response for the accessed Servlet or JSP component deployed on the Web server.

Slide 26

Let us understand Web Container.

Web Container

- ❖ Manages execution of Servlets and JSP pages.
- ❖ Takes request from a Web server and passes it to a servlet for processing.
- ❖ Manages the servlet life cycle and other services such as security, transaction, and so on for the Web components.
- ❖ Referred as Servlet container.
- ❖ Figure depicts a Web container.



- ❖ The performance of a servlet depends upon the efficiency of the Web container.

© Aptech Ltd. Web Component Development Using Java/Session 1 26

Use slide 26 to explain Web container. Tell them that The Web container is a program that manages execution of Servlets and JSP pages. . The Web container takes request from a Web server and passes it to a servlet for processing. It manages the Servlet life cycle and other services such as security, transaction, database access, and so on for the Web components. The performance of a servlet depends upon the efficiency of the Web container. The Web container is also referred as Servlet container.

Use the figure in slide 26 to show them how Servlet threads are managed by a Web container.

Tips:

Apache Tomcat is a widely used implementation of the Java Servlet Specification, which has been developed as an open-source project by the Apache Software Foundation (ASF) since 1999, when the project source was donated to the ASF by Sun Microsystems. Tomcat is actually composed of a number of components, including a Tomcat JSP engine and a variety of different connectors, but its core component is the Servlet container called Catalina. Catalina provides Tomcat's actual implementation of the Servlet specification; when you start up your Tomcat server, you're actually starting Catalina.

The configuration files of Catalina found in the Tomcat installed directory on the system are as follows:

- catalina.policy
- catalina.properties
- logging.properties
- content.xml
- server.xml
- tomcat-users.xml
- web.xml

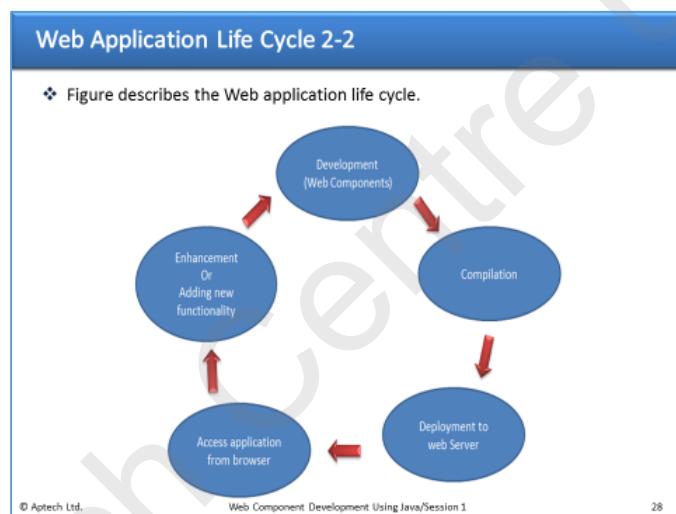
Slides 27 and 28

Let us understand Web application life cycle.

Web Application Life Cycle 1-2

- ❖ Develop the Web component code.
- ❖ Develop the Web application deployment descriptor.
- ❖ Compile the Web application components and helper classes referenced by the components.
- ❖ Optionally, package the application into a deployable unit.
- ❖ Deploy the application into a Web container.
- ❖ Access a URL which references the Web application.

© Aptech Ltd. Web Component Development Using Java/Session 1 27



Use slides 27 and 28 to explain Web application life cycle. Tell them that Web applications are needed to create a dynamic application using different Web components such as Java Servlets, JSP, HTML, Figures, and JavaScript helper classes and libraries.

Use slide 27 to summarize the process or life cycle for creating, deploying, and executing a Web application. Tell them that the process or life cycle for creating, deploying, and executing a Web application can be summarized as follows:

- Develop the Web component code.
- Develop the Web application deployment descriptor.
- Compile the Web application components and helper classes referenced by the components.
- Optionally package the application into a deployable unit.
- Deploy the application into a Web container.
- Access a URL which references the Web application.

Then, use the figure in slide 28 to demonstrate the Web application life cycle.

Slide 29

Let us understand packaging of Web applications.

Packaging Web Applications

- ❖ A Web application is composed of Web pages and Web components which are collectively referred as Web resources.
- ❖ The Web applications also need to be packaged with Web resources before the application is deployed on the Web server.
- ❖ The package file used for a Web application has a specific set of directory structure:
 - ❑ That is packaged in the `.war` file.
 - ❑ Helps the Web container to ensure proper functionality of the application.

© Aptech Ltd. Web Component Development Using Java/Session 1 29

Use slide 29 to explain packaging of Web applications. Tell them that a Web application is composed of Web pages and Web components which are collectively referred as Web resources. These files are to be placed in the correct structure, so that the files are located properly by the Web engine for correct functionality.

In addition to Web components, a Web application archive can contain other files including the following:

- Server-side utility classes (database beans, shopping cart classes, and so on). Often these classes conform to the JavaBeans component architecture.
- Static Web presentation content (HTML, figure, sound files, and so on).
- Client-side classes (applets and utility classes).

Java applications are packaged as Java Archive (JAR) which contains multiple classes and other related resources. All the components required in the Java application are packaged into a single `.jar` file. Similar to Java applications, the Web applications also need to be packaged with Web resources before the application is deployed on the Web server.

The package file used for a Web application has a specific set of directory structure which helps the Web container to ensure proper functionality of the application. The Web applications are therefore, packaged in a Web Archive (WAR) which distinguishes it from the normal Java application. The Web application resources are packaged in the `.war` file.

Tips:

A Web application can run from a War file deployed on the Web server or you can run the pages directly from the project directory, created on the structure of War file. The top-level directory of a War file is called the *document root* of the application.

Slide 30

Let us understand Web application context root.

Web Application Context Root

- ❖ Each Web application is identified or assigned with a **context root**.
- ❖ The context root or context path of the Web application:
 - Is the base URL used for locating Web pages and Web components such as Servlet and JSP within the application.
- ❖ Example:
 - Suppose the application is deployed on the local Web server.
 - Then, the URL to access the `index.html` page will be:
`http://localhost:8080/WebApp/page/index.html`.
 - Thus, the context root of the deployed application is the `/WebApp`.

© Aptech Ltd. Web Component Development Using Java/Session 1 30

Use slide 30 to explain Web application context root. Tell them that the Web container supports multiple Web applications, so each application is identified or assigned with a context root. The context root is specified when the Web module is deployed, and it should begin with a forward slash (/) and finish with a string.

For example, consider a `.war` file named `WebApp` contains a file named `index.html` in a `page` folder. When the `.war` file is deployed on the Java Web server, then you need the URL to access the pages from the application. Suppose the application is deployed on the local Web server, then the URL to access the `index.html` page will be
`http://localhost:8080/WebApp/page/index.html`. Thus, the context root of the deployed application is the `/WebApp`.

Slide 31

Let us understand Web application directory structure.

Web Application Directory Structure

- ❖ The context root of the Web application contains two main components in the directory structure:
 - **Static files** - All the HTML pages and images comprise the static files.
Images can be collectively stored in an images directory.
 - **WEB-INF** - Exists within the context root, cannot be referenced.
- ❖ Figure shows the directory structure of the .war file.

```

graph TD
    webapps[webapps/] --> exampleapp[exampleapp/]
    exampleapp --> index[index.jsp]
    exampleapp --> contact[contact.html]
    exampleapp --> images[images/]
    images --> photo[photo.jpg]
    exampleapp --> META_INF[META-INF/]
    META_INF --> webFragment[web-fragment.xml]
    exampleapp --> WEB_INF[WEB-INF/]
    WEB_INF --> webXml[web.xml]
    WEB_INF --> classes[classes/]
    classes --> sampleServlet[SampleServlet.class]
    WEB_INF --> lib[lib/]
    lib --> jar[jar]
    WEB_INF --> tags[tags/]
  
```

© Aptech Ltd. Web Component Development Using Java/Session 1 31

Use slide 31 to explain Web application directory structure. Tell them that the contents of the Web application are accessed from the application's context root. Web applications use a standard directory structure defined in the Java EE specification. A Web application can be deployed as a collection of files that use this directory structure or as an archived file called a WAR file. Explain the two main components of the directory structure with examples.

For Static files, tell them, for example, to access a file `index.html` in the book folder on the Website `www.corejava.com`, the URL would be `http://www.java.com/book/index.html`.

For WEB-INF files, tell them, for example, referencing `www.java.com/book/WEB-INF` is not possible. Files contained in this directory cannot be served directly to a client.

Use the figure in slide 31 to explain the directory structure of the .war file.

Slides 32 and 33

Let us understand WEB-INF directory.

WEB-INF Directory 1-2

The contents of WEB-INF directory are as follows:

- ❖ **classes directory**
 - Servlet classes, JavaBean classes, and any other classes required by the Web applications are stored in this directory.
 - Example: If a servlet or JavaBean class is part of a package such as `www.bookstore.com`, then the corresponding class file will be placed in `classes/com/bookstore` directory.
- ❖ **web.xml**
 - Basically, it is an XML structured file which provides configuration information for the components of the Web application.
 - Called as the deployment descriptor of the Web application.
 - Includes information such as the default pages to show mapping Servlets with their URLs, and so on.



© Aptech Ltd. Web Component Development Using Java/Session 1 32

WEB-INF Directory 2-2

- ❖ **lib directory**
 - Library JAR files used by the application are stored within the WEB-INF/lib directory.
 - Example: Database driver files.
- ❖ **tags directory**
 - Contains tag files, which provide implementation for custom tags.
- ❖ **tag library descriptor (tld) files**
 - Provide information about the attributes of a custom tag and the class to be invoked when the tag comes across in a JSP page.
 - Files have .tld extension.

© Aptech Ltd. Web Component Development Using Java/Session 1 33

Use slides 32 and 33 to explain WEB-INF directory structure. In classes directory, tell them that Servlets and JavaBean classes that are part of a package, are placed to match their package names.

In lib directory, tell them that lib directory contains .java archive files required for Web application such as database drivers. In tags directory, tell them that tag files have .tag extension. In tag library descriptor files, tell them that tag library descriptor files are required only if tag files need to be distributed as a separate package or if the custom tags are implemented using Java.

In-Class Question:

After you finish explaining WEB-INF directory, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which file is called the deployment descriptor of the Web application?

Answer:

web.xml file.

Slides 34 to 37

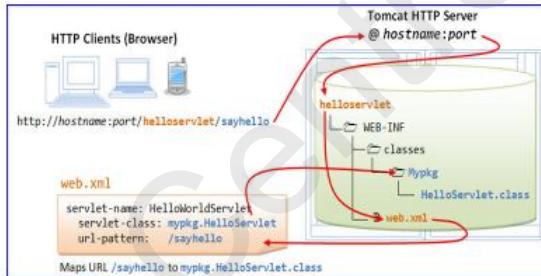
Let us understand deployment descriptor.

Deployment Descriptor 1-4

- ❖ A configuration file which describes how the Web application should be deployed.
- ❖ Written using XML with name `web.xml` and placed under the **WEB-INF** folder, **WEB-INF/web.xml**.
- ❖ When the Web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the Servlet that handles the request.
- ❖ **Example:** To map a URL request to a Servlet, the following settings must be done in the `web.xml` file:
 - ❑ Declare the servlet with the `<servlet>` element.
 - ❑ Define a mapping from a URL path to a Servlet declaration with the `<servlet-mapping>` element.

Deployment Descriptor 3-4

- ❖ Figure shows the mapping of Servlet with the URL through which it is accessible to the application.



Deployment Descriptor 2-4

- ❖ The code snippet shows the configuration setting for the `HelloServlet` class created in the Web application.

```

<!-- web.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<!--
<servlet>
  <servlet-name>HelloWorldServlet</servlet-name>
  <servlet-class>mypkg.HelloServlet</servlet-
class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloWorldServlet</servlet-name>
  <url-pattern>/sayhello</url-pattern>
</servlet-mapping>
</web-app>

```

Deployment Descriptor 4-4

- ❖ Servlet 3.0 has also introduced a new feature called **Web fragments**.
- ❖ Web fragment is the mechanism that:
 - ❑ Include deployment descriptor information which is specific to a library.
 - ❑ Helps to segregate the unrelated information from the main deployment information.
 - ❑ Is stored in the folder META-INF\web-fragment.xml of the Web application directory structure.

Use slides 34 to 37 to explain deployment descriptor.

In slide 34, after giving the example, tell them that the `<servlet>` element declares the Servlet, which includes a name used to refer to the Servlet by other elements in deployment descriptor and the Servlet class. The name for each Servlet must be unique across the deployment descriptor. The `<servlet-mapping>` element specifies a URL pattern and the name of a declared Servlet (declared in `<servlet>` element) to use for requests whose URL matches the pattern.

Use the code snippet in slide 35 to display the configuration setting for the `HelloServlet` class created in the Web application named `FirstWebApplication`. Tell them that, in the code, the URL pattern to access the `HelloServlet` class will be

`http://localhost:8080/FirstWebApplication/sayhello.`

Use the figure in slide 36 to demonstrate the mapping of Servlet with the URL through which it is accessible to the application.

Proceed with the explanation of deployment descriptor using slide 37. Tell them that from Java EE 6 onwards, the `web.xml` file is optional, as the configuration information can be applied with the annotations. Annotations are meta information embedded in the Java source files. The meta information creates the dynamic configuration setting for the deployed component during runtime. Then, explain students about web fragments introduced in Servlet 3.0. It allows the Web developer to modularize the deployment descriptors. This means that there can be multiple Web fragments, each representing a logical segments. The logical segments can be referred forming a complete `web.xml` file.

Consider an example to register a Servlet and its listener in a `web-fragment.xml` file:

```
<web-fragment>
  <servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>com.MyServlet</servlet-class>
  </servlet>
  <listener>
```

```

<listener-class>myServlet.myServletListener</listener-class>
</listener>
</web-fragment>

```

A web fragment must be in a file named `web-fragment.xml` and can be placed in any location in a web application's classpath. However, it is typically placed in the `META-INF` directory of the JAR file, which will typically reside in the `WEB-INF/lib` directory of the Web application.

This way multiple web-fragments can be created and their ordering can be specified in which the deployment descriptors can be processed by the Web engine.

Tips:

Servlet 3.0 supports absolute ordering and relative ordering of deployment descriptors. The developer can specify absolute ordering using the `<absolute-ordering>` element in the `web.xml` file. The developer can specify relative ordering with an `<ordering>` element in the `web-fragment.xml` file.

Consider an example where there are two Web fragments created namely `fragment1` and `fragment2`. So, these can be ordered in `web.xml` file based on absolute or relative ordering of the processing.

```

<web-app>
    <name>MyApp</name>
    <absolute-ordering>
        <name>Fragment1</name>
        <name>Fragment2</name>
    </absolute-ordering>
    ...
</web-app>

```

Here, the processing order would be as follows:

- `web.xml` which is always processed first.
- Fragment1
- Fragment2

After explaining Web fragments, tell them that a developer might have created functionality for the application separately in some other Java EE project and import its `.jar` in `WEB-INF\lib` as plug-in within the Web application.

In addition, tell them that the content of `web.xml` and `web-fragment.xml` are almost the same, but the difference is in their location, we put `web.xml` in `WEB-INF\web.xml` and `web-fragment.xml` in `METAINF\web-fragment.xml` in the Web archive file.

Slide 38

Let us understand NetBeans IDE.

Developing Web Applications in NetBeans IDE

- ❖ NetBeans Integrated Development Environment (IDE) is used to create Java EE based projects with JSP's and Servlets.
- ❖ Some of the features of the NetBeans IDE are as follows:
 - ❑ A source code editor which is similar to an HTML text editor.
 - ❑ A compiler compiles the source code into an executable program and an interpreter runs programs and scripts that do not need to be compiled.
 - ❑ Build automation tools help automate the processes that need to happen with most software developments such as debugging, compiling, and deployment.
 - ❑ A debugger helps pin-point the exact spot of a problem or error in the source code.

© Aptech Ltd. Web Component Development Using Java/Session 1 38

Use slide 38 to explain NetBeans Integrated Development Environment (IDE). Tell them that the developer can create and work with Web applications using different IDE and tools. Mainly, NetBeans IDE is used to create Java EE based projects with JSP's and Servlets.

NetBeans IDE is a software program which is designed to help programmers and developers build and test Java applications. NetBeans IDE provides text or source code editor, debugger, and compiler all in one in a single environment. NetBeans IDE is written in Java and can run on various platforms supporting a compatible Java Virtual Machine (JVM). Applications based on this IDE can be extended by third-party developers.

Slides 39 to 41

Let us understand how to create a new Web project in NetBeans IDE.

Creating a New Web Project in NetBeans IDE

- ❖ Open NetBeans IDE and select **New Project** from the **File** menu.
- ❖ Select **Java Web** from the **Categories** list.
- ❖ Select **Web Application** from the **Projects** list.
- ❖ Click **Next** to specify the name and location.
- ❖ Figure shows how to create a project.

© Aptech Ltd. Web Component Development Using Java/Session 1 39

Specifying Name and Location for the Project 1-2

- ❖ Enter the name of the project, for example, **FirstWebApp**.
- ❖ Click the **Browse** button and select a location for saving the project.
- ❖ Click **Next** to specify the server used for deploying Web application.
- ❖ Click **Server** drop-down list and select the Web server for the application as shown in the figure.

© Aptech Ltd. Web Component Development Using Java/Session 1 40

Specifying Name and Location for the Project 2-2

- ❖ Click **Next** and select frameworks used in Web application. Currently, the Web application is not using any framework.
- ❖ Click **Finish**.
- ❖ Figure shows the structure of the **FirstWebApp** project created in NetBeans IDE.

© Aptech Ltd. Web Component Development Using Java/Session 1 41

Use slides 39 to 41 to explain the process of creating a new Web project in NetBeans IDE.

Use slide 39 to explain the steps to create a new Web application in NetBeans IDE.

Use slides 40 and 41 to explain how to specify name, for example, **FirstWebApp**, and location for a new project. After explaining the fourth step, mention that If Web server is not available, then you can add a new one (for current application we will be using Apache Tomcat 7.0).

Use the figure in slide 40 to demonstrate how to select a Web server for the project.

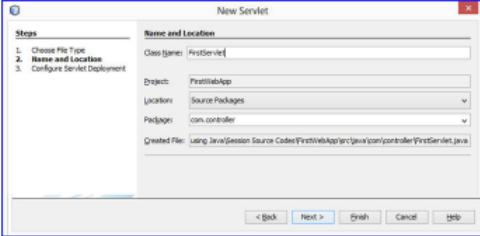
Use the figure in slide 41 to display the structure of the **FirstWebApp** project created in NetBeans IDE.

Slides 42 to 44

Let us understand how to write the code for Servlet.

Writing the Code for Servlet 1-3

- ❖ Right-click the **Source Packages** and select **Servlet** from the context menu. This opens **New Servlet** window.
- ❖ Type the name of the Servlet as **FirstServlet** and package as **com.controller** as shown in the following figure.



□ Click the **Finish** button.

© Aptech Ltd. Web Component Development Using Java/Session 1 42

Writing the Code for Servlet 2-3

- ❖ The code snippet shows the code generated for the 'FirstServlet' class.

```
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet {
protected void processRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet FirstServlet</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet FirstServlet at " + request.getContextPath() + "</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

- ❖ The **out** object of **PrintWriter** class is created to display the message on the Web page in response.

© Aptech Ltd. Web Component Development Using Java/Session 1 43

Writing the Code for Servlet 3-3

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    processRequest(request, response);
}
...
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    processRequest(request, response);
}
```

- ❖ The **processRequest ()** method in NetBeans IDE contains the default generated Servlet code.
- ❖ This method is invoked by **doGet ()/doPost ()** method based on how the data is sent by the client in HTTP request.
- ❖ Change the **out.println("<h1>Servlet FirstServlet at " + request.getContextPath() + "</h1>");** with this **out.println("<h1> Hello World </ h1>");**

© Aptech Ltd. Web Component Development Using Java/Session 1 44

Use slides 42 to 44 to explain how to write the code for Servlet.

Use the figure in slide 42 to demonstrate typing of the name of the Servlet as **FirstServlet** and package as **com.controller**. Tell them that a new Java class in the package needs to be created. For example, a class named as ‘FirstServlet’ is created within the package com.controller.

Use the code snippet in slides 43 and 44 to display the code generated for the ‘FirstServlet’ class. Tell them that as shown in the given code snippet, the `out` object of `PrintWriter` class is created within the `processRequest()` method in NetBean IDE. This method is invoked by `doGet()` and `doPost()` method based on how the data is sent by the client in HTTP request. The `println()` method of `out` object is used to display the message string on the Web page. Change the `out.println("<h1>Servlet FirstServlet at " + request.getContextPath() + "</h1>");` with this `out.println("<h1> Hello World </h1>");`

Slide 45

Let us learn how to create the deployment descriptor.

Creating the Deployment Descriptor

The code snippet shows the `web.xml` file that is updated with the configuration settings of the `FirstServlet`.

```
<?xml version="1.0" encoding="UTF-8"?>
<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>com.controller.FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/myservlet</url-pattern>
</servlet-mapping>
</web-app>
```

© Aptech Ltd. Web Component Development Using Java/Session 1 45

Use slide 45 to explain how to create the deployment descriptor. Use the code snippet to display the `web.xml` file within the `Web-INF` directory.

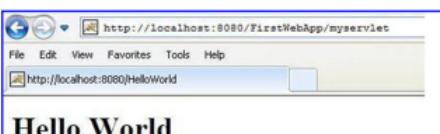
Tell them that the `web.xml` file is updated with the configuration settings of the `FirstServlet`. The Servlet class will be accessed through the URL, `/myservlet`.

Slide 46

Let us learn how to build and execute the Web application.

Build and Execute the Web Application

- ❖ To run the application, right-click the project and select **Run** from the context menu.
- ❖ When you run the application in NetBeans IDE:
 - It first compiles the Servlet code and then deploys the resource to the Web server.
 - If Web server is not running, first start the server and then deploy the Web application to server.
 - After building and deployment is completed successfully, NetBeans will launch the Servlet page in the browser as shown in figure.



The screenshot shows a web browser window with the URL `http://localhost:8080/FirstWebApp/myservlet` in the address bar. The page content displays the text "Hello World".

© Aptech Ltd. Web Component Development Using Java/Session 1 46

Use slide 46 to explain the steps used to build and execute the Web application. Use the figure in slide 46 to demonstrate how NetBeans launches the Servlet page in the browser.

Tell them when you run the application, it first compile the Servlet code and then deploy the resource to the Web server, if Web server is not running, first start the server and then deploy the Web application to server. After building and deployment is completed successfully, NetBeans will launch the Servlet page in the browser as shown in the slide.

Slide 47

Let us summarize the session.

Summary

- ❖ An application is a collection of programs designed to perform a particular task. Different types of applications are designed for different purposes.
- ❖ A Web application is a software application that runs on a Web server. The Web application basically has three components: the presentation layer, the application layer, also known as business layer, and the data access layer.
- ❖ The most common technologies for communication on the Web are HTML and HTTP. HTML is a presentation language which enables Web designer to create visually attractive Web pages.
- ❖ The requests and responses sent to a Web application from one computer to another computer are sent using HTTP. HTTP does not store any connection information about the page and hence, it is referred to as a stateless protocol.
- ❖ The HTTP request messages uses the HTTP methods for transmitting request data over the Web.
- ❖ Java Web application comprises servlets, JSP pages, images, HTML files, JavaBeans, Applets, and Java classes.
- ❖ The Web applications are packaged in the .war file which allows the content of the Web application are accessed from the application's context root.
- ❖ A deployment descriptor describes how the Web application should be deployed. It is written using XML with name web.xml and placed under the WEB-INF folder, that is, WEB-INF/web.xml.
- ❖ Java Web applications are developed in NetBeans IDE which provides fully-integrated environment for building and executing the Web applications.

© Aptech Ltd. Web Component Development Using Java/Session 1 47

In slide 47, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

1.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore servlet architecture, life cycle of servlet, use of response headers, and error handling in servlet.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 2 – Java Servlet

2.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

2.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the Servlet API
- Explain the servlet architecture and life cycle of Servlet
- Explain the methods of ServletRequest and HttpServletRequest interfaces
- Explain the methods of ServletResponse and HttpServletResponse interfaces
- Describe the use of response headers
- Explain how to read text and binary data from a request
- Explain the ServletConfig and ServletContext interface
- Explain redirection of client requests
- Explain RequestDispatcher interface
- Explain error handling in Servlet

2.1.2 Teaching Skills

To teach this session successfully, you should be aware with the Java Servlet API. You should have thorough understanding of Servlet architecture and its life cycle methods. You should also prepare yourself with the implementation of various Servlet API classes such as request and response objects and Servlet initialization parameters.

You must prepare himself with the concept of redirection and redirecting request object using RequestDispatcher interface. Finally, the understanding of error handling mechanism in Servlets is expected.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives
<ul style="list-style-type: none">❖ Explain the Servlet API❖ Explain the servlet architecture and life cycle of Servlet❖ Explain the methods of ServletRequest and HttpServletRequest interfaces❖ Explain the methods of ServletResponse and HttpServletResponse interfaces❖ Describe the use of response headers❖ Explain how to read text and binary data from a request❖ Explain the ServletConfig and ServletContext interface❖ Explain redirection of client requests❖ Explain RequestDispatcher interface❖ Explain error handling in Servlet

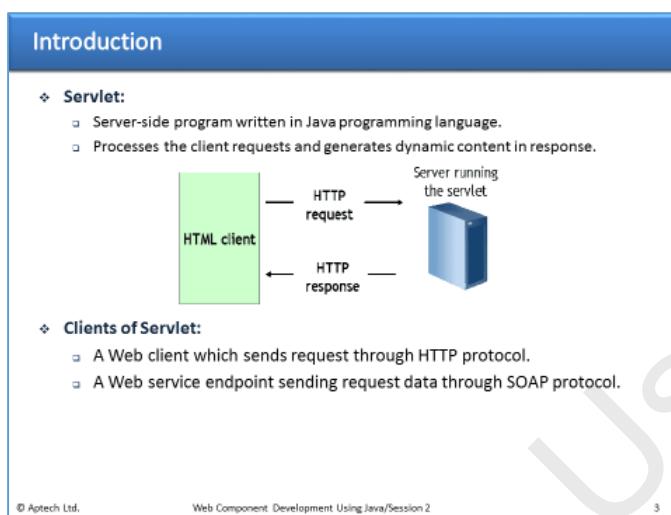
Tell them that they will be introduced to Java Servlet and its architecture. They will be taught about Servlet API and also the life cycle of a Servlet. In addition, methods of ServletRequest and ServletResponse are also explained in detail.

Further, tell them that they will be introduced with the working of ServletConfig and Servlet Context interfaces. Finally, the session concludes with the explanation about RequestDispatcher interface and error handling mechanism in Servlet.

2.2 In-Class Explanations

Slide 3

Let us understand Servlet.



Use slide 3 to introduce Servlet.

Brief the students that a Servlet is a server-side program written in Java programming language. It is based on component-based framework where the Servlet is designed as a Web component to be deployed on Java compatible Web servers. As Servlet is a Web components, it has to be accessed through HTTP protocol. However, Servlet can be accessed through other protocols also.

Tell them that a Java Servlet is similar to a CGI program as it runs on a Web server and responds to HTTP server. However, there is a difference in their executing architectures. The Servlet runs as a thread in the Web container, however, earlier CGI scripts were executed as separate process.

Tips:

The Web container is an operating system process running as a service within the Java Web server.

Then, use the figure given on slide 3 to explain the functioning of a Servlet. Servlet processes the client request, produces contents, generates response, and sends it back directly to the Web engine which in turn interacts with the clients.

The different types of clients to access a Servlet are:

- Web client – Can be HTML or JSP page which can access the Servlet through HTTP protocol.
- Web service endpoint – Interacts with a Servlet using SOAP protocol.

Slides 4 to 8

Let us understand Servlet API.

Servlet API 1-5

- ❖ All the classes related to developing and managing servlets are provided in two packages, they are as follows:
 - javax.servlet**
 - It contains generic interfaces and classes that are implemented and extended by all servlets.
 - It provides a framework for the Servlet operations.
 - It includes an interface named `Servlet` which defines the life cycle methods for a servlet.
 - Every servlet must implement the `javax.servlet.Servlet` interface directly or indirectly.
 - javax.servlet.http**
 - It contains classes and interfaces used for developing a servlet that works with HTTP protocol.

© Aptech Ltd.

Web Component Development Using Java/Session 2

4

Servlet API 2-5

- ❖ Figure displays the Servlet API hierarchy.

```

graph TD
    Object[Object] --> GenericServlet[GenericServlet]
    GenericServlet --> HttpServlet[HttpServlet]
    HttpServlet --> UserDefinedServlet[User-defined Servlet]
    UserDefinedServlet --> Object

    subgraph Interfaces [Interfaces]
        Servlet
        ServletConfig
        Serializable
    end

    Object -- "Base Class" --> GenericServlet
    GenericServlet -- "Built-in Servlet Class" --> HttpServlet
    HttpServlet -- "User-defined Servlet Class" --> UserDefinedServlet

    Serializable -- "Implements" --> GenericServlet
    Serializable -- "Implements" --> HttpServlet
  
```

The diagram illustrates the class hierarchy of the Servlet API. At the top left, there is a box labeled 'Interfaces' containing three items: `Servlet`, `ServletConfig`, and `Serializable`. To the right, a vertical stack of classes is shown: `Object` (labeled 'Base Class'), `GenericServlet` (labeled 'Built-in Servlet Class'), `HttpServlet` (labeled 'User-defined Servlet Class'), and finally `User-defined Servlet` (also labeled 'User-defined Servlet Class'). Solid arrows point from `Object` up to `GenericServlet`, from `GenericServlet` up to `HttpServlet`, and from `HttpServlet` up to `User-defined Servlet`. Dashed blue arrows labeled 'Implements' point from both `Serializable` interfaces to `GenericServlet` and `HttpServlet`.

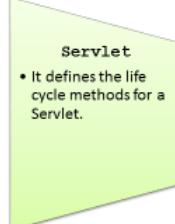
© Aptech Ltd.

Web Component Development Using Java/Session 2

5

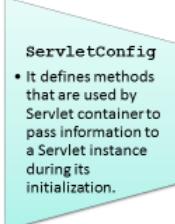
Servlet API 3-5

- ❖ The `javax.servlet` package contains an abstract class named `GenericServlet`.
- ❖ The `GenericServlet` inherits from `Object` class and helps to design a protocol independent servlet.
- ❖ The `GenericServlet` implements three interfaces, they are as follows:



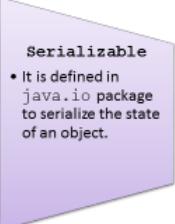
Servlet

- It defines the life cycle methods for a Servlet.



ServletConfig

- It defines methods that are used by Servlet container to pass information to a Servlet instance during its initialization.



Serializable

- It is defined in `java.io` package to serialize the state of an object.

© Aptech Ltd. Web Component Development Using Java/Session 2

6

Servlet API 4-5

- ❖ The code snippet demonstrates how to create a protocol-independent login servlet.

```
public class LoginServlet extends GenericServlet{
    ...
}
```

- ❖ In the given code, the class named `LoginServlet` is a generic servlet.
- ❖ To develop a servlet which is accessible only through a Web browser, user can create an HTTP Servlet by extending `HttpServlet` class.
- ❖ The `HttpServlet` class is a subclass of `GenericServlet` and enables to create an HTTP-based Servlet as part of a Web application.

© Aptech Ltd. Web Component Development Using Java/Session 2

7

Servlet API 5-5

- ❖ The code snippet shows the creation of login servlet to be accessed on the Web.

```
public class LoginServlet extends HttpServlet{
    ...
}
```

- ❖ In the given code, the `LoginServlet` is defined as HTTP servlet, so it will be accessed through HTTP protocol.

© Aptech Ltd. Web Component Development Using Java/Session 2

8

Use slides 4 to 8 to explain Servlet API. Tell them that all the classes related to developing and managing Servlets are provided in two packages namely, `javax.servlet` and `javax.servlet.http`.

The `javax.servlet` package classes and interfaces are not specific to any protocol. The `javax.servlet` package contains a number of classes and interfaces that define the interface between the Servlet class and the JVM. JVM provides the instances of Servlet which are based on the Java standards which make it compatible with the Servlet container.

Use the figure given on slide 5 to explain the Servlet API hierarchy. Tell them that the Servlet hierarchy includes two top-level interfaces. These interfaces are namely, `java.servlet.Servlet` and `javax.servlet.ServletConfig`.

Tell them that every Servlet must implement the `javax.servlet.Servlet` interface directly or indirectly by extending the class which already implements this interface. The `Servlet` interface is the central abstraction of the Servlet API.

Using slide 6, explain that the `javax.servlet` package contains an abstract class named `GenericServlet` which inherits from `Object` class and helps to design a protocol independent Servlet. You must inherit the

`GenericServlet` class to design your own Servlet. The `GenericServlet` implements three interfaces namely,

- `Servlet` – This interface defines the life cycle methods for a Servlet.
- `ServletConfig` – This interface defines methods that are used by Servlet container to pass information to a Servlet instance during its initialization.
- `Serializable` – This interface is defined in `java.io` package to serialize the state of an object.

Use the code snippet given on slide 7 to demonstrate how to create a protocol-independent Login Servlet for a situation where you need to develop a Login Servlet to validate the user's information.

Then, tell them that the `GenericServlet` class is extended by the `HttpServlet` class to develop a Servlet which is accessible only through a Web browser. The developer can extend the user-defined Servlet class from `HttpServlet`. The `HttpServlet` class is a subclass of `GenericServlet` and enables to create an HTTP-based Servlet as part of a Web application.

Use the code snippet given on slide 8 to display the creation of LoginServlet to be accessed on the Web.

In-Class Question:

After you finish explaining Servlet API, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



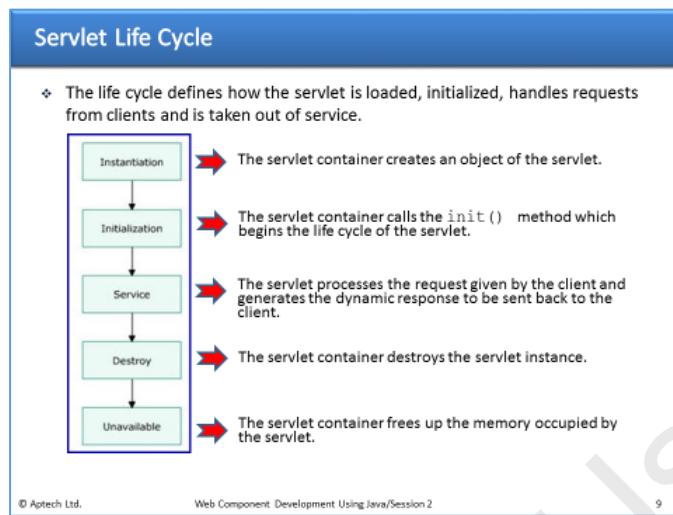
Which are the two packages in which all the classes related to developing and managing Servlets are included?

Answer:

`javax.servlet` and `javax.servlet.http`

Slide 9

Let us understand Servlet life cycle.



Use slide 9 to explain the life cycle of a Servlet. Tell them that the container in which the servlet has been deployed controls the life cycle of the Servlet. The life cycle defines how the servlet is loaded, initialized, handles requests from clients and is taken out of service. In the **Instantiation** stage, the Servlet container creates an object of the servlet. When the Servlet is instantiated, it is in **initialization** stage. After initialization, the Servlet is ready to service the client's request. A Servlet can come to the **Destroy** stage of the life cycle, when the Servlet instances is destroyed, explicit request from administrator to destroy the Servlet instance, Web server shutting down, or a Servlet instance is idle and not processing any request.

Slide 10

Let us understand Servlet life cycle methods.

Servlet Life Cycle Methods

- ❖ The servlet life cycle is defined by the following methods:
 - init()** Initializes the servlet
 - The life cycle of the servlet begins with the `init()` method.
 - This method is called only once by the server and not again for each user request.

- service()** Processes the request made by the client

 - It passes the `ServletRequest` and `ServletResponse` objects which collect and pass information on which the request was received.

- destroy()** Destroys the servlet interface, if there are no more requests to be processed

 - A server calls this method after all requests have been processed or a server-specific number of seconds have passed, whichever comes first.

© Aptech Ltd.

Web Component Development Using Java/Session 2

10

Use slide 10 to explain Servlet life cycle methods.

Tell them that `init` method is called when the Servlet is first created and is used for one-time initializations. The Servlet is generally generated when a user first invokes a URL corresponding to the servlet. You can also specify the Servlet to be loaded when the server is first started.

The `service()` method is the main method to accomplish the actual job. The servlet container requests the `service()` method to handle requests coming from the browsers and to generate the response to be sent back to the client. Whenever, the server receives a request for Servlet execution, it creates a new thread and services the request. The `service()` method then checks the HTTP request type and calls the appropriate methods.

The `destroy()` method is called only at the end of the Servlet life cycle of a Servlet. The Servlet can end database connections, stop background threads, compose cookie lists, and accomplish any cleanup activities. Once the `destroy()` method is requested the Servlet object is marked for garbage collection.

In-Class Question:

After you finish explaining Servlet life cycle methods, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which method is used for one-time initializations of Servlets?

Answer:

`init` method.

Slides 11 and 12

Let us understand Servlet architecture.

Servlet Architecture 1-2

- ❖ The servlet container is responsible for handling servlet execution which is based on multithreaded model.
- ❖ Figure shows the handling of multiple clients request by a servlet.

```

graph LR
    WB1[Web browser] -- "Request A: http://localhost:8080/.../LoginServlet" --> WE[Web Engine]
    WB2[Web browser] -- "Request B: http://localhost:8080/.../LoginServlet" --> WE
    WE --> SC[Servlet Container]
    SC -- "Request A" --> TA[Thread A]
    SC -- "Request B" --> TB[Thread B]
    TA --> S[Servlet]
    TB --> S
    S -- "init()" --> S
    S -- "service()" --> S
    S -- "destroy()" --> S
    
```

© Aptech Ltd. Web Component Development Using Java/Session 2 11

Servlet Architecture 2-2

- ❖ The flow of information in a servlet is as follows:

The Web browser sends an HTTP request to the Web server through a Uniform Resource Locator (URL)

On receiving the request, the Web engine looks into the web.xml deployment descriptor file to match the URL with the registered Servlets

Once the requested Servlet is found, the Web engine forwards it to the Servlet container which is responsible for instantiating the Servlet instance

The container first locates the servlet class, loads the servlet, and creates an instance of the servlet in the memory

After the Servlet is loaded by the Servlet container, it is initialized completely and then the client request is processed

© Aptech Ltd. Web Component Development Using Java/Session 2 12

Use slides 11 and 12 to explain Servlet architecture.

Tell the students that the servlet container is responsible for handling Servlet execution which is based on multithreaded model. This means when the container receives the request for the same Servlet, it creates a new thread that handles the execution of the Servlet and dies once the execution is completed.

Use the figure given on slide 11 to demonstrate the flow of information in a Servlet.

Use slide 12 to describe the flow of information in a Servlet. In the first step, give an example of a URL as: <http://localhost:8080/.../LoginServlet>.

Tips:

To service multiple clients' request, the servlet container creates multiple threads for the same Servlet instance. Each instance handles request received from the client and generates response that is sent to the Web engine which in turn, sends the response back to the client.

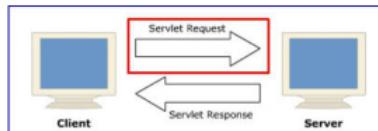
©Aptech Limited

Slide 13

Let us understand handling Servlet request.

Handling Servlet Request

- ❖ The `ServletRequest` interface:
 - Retrieves client-specific request parameters such as data entered in online forms.
 - Provides access to the specific information required to process the request appropriately.
 - Handles a request, the server passes a request object, which implements the `ServletRequest` interface.
 - Has two access parts of the raw request such as headers and input stream.
- ❖ Figure depicts a Servlet request.



Use slide 13 to explain Servlet request handling.

Tell the students that when a Servlet is requested to handle a request, it needs specific information about the request so, that it can process the request appropriately. When a Servlet handles a request, the server passes a request object, which implements the `ServletRequest` interface. With the `ServletRequest` object, the Servlet can find out information about the actual request such as protocol, Uniform Resource Locator (URL), and type. It can access parts of the raw request such as headers and input stream and also retrieve client-specific request parameters, such as data entered in online forms.

Use the figure given on slide 13 to explain the passing of `ServletRequest` object between the client and the Web server.

Slides 14 to 16

Let us understand `ServletRequest` interface.

ServletRequest Interface 1-3

- The `ServletRequest` object is passed as an argument to the `service()` method of the servlet.
- The methods defined in the `ServletRequest` interface are as follows:

 `public String getParameter(String s)`

The method returns the value of a specified parameter that is sent along with the request information.

 `public String getParameter(String name)`

The method retrieves the value of an attribute specified by name, that was set using the `setAttribute()` method and returns null when no attribute with the specified name exists.

© Aptech Ltd.

Web Component Development Using Java/Session 2

14

ServletRequest Interface 2-3

 `public int getContentLength()`

The method returns the length of content in bytes and returns -1, if the length is not known.

- The code snippet shows the use of `getContentLength()` method.

```
/* The length of the content is passed to len1 and
compared with the size of file limit */
if((len1 = request.getContentLength()) > fileLimit){
System.out.println ("Name1 "+login +" Id1 " +id +
"Expected Upload size is more than specified file
Limit");
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 2

15

ServletRequest Interface 3-3

 `public ServletInputStream getInputStream() throws
IOException`

The method returns the binary data of the body of request, requested by the client and stores it in a `ServletInputStream` object.

 `ServletInputStream inStream = request.getInputStream();
public String getServerName()`

The method returns the host name of the server to which the client request was sent.

© Aptech Ltd.

Web Component Development Using Java/Session 2

16

Use slides 14 to 16 to explain `ServletRequest` interface.

Tell them that the `ServletRequest` object provides request information, such as parameter values and names, attributes, content length and type, header information, and so on to the servlets.

Some of the methods defined in the `ServletRequest` interface are as follows:

- `public String getParameter(String s)` method - The statement, `String name1 = request.getParameter("names1");` retrieves the value from the parameter `names1` passed in the request.
- `public Object getAttribute(String name)` method - `Object cobj1 = cnt1.getAttribute("obj1");`. The method retrieves the value of an attribute specified by name.
- `public int getContentLength()` - Returns the length of content in bytes and returns -1, if the length is not known.

Using the code snippet given on slide 15 to explain how to use the `getContentLength()` method to check the length of the content sent in the request. If the file size is more than the expected limit, then the appropriate message is displayed on the page.

- `public ServletInputStream getInputStream() throws IOException` method – Is used to read the binary data from the message body sent in the request. The following statement can be used, `ServletInputStream inStream1 = request.getInputStream();` to read the binary data from the request.
- `public String getServerName()` method - `String serverName = request.getServerName();`. The statement returns the host name of the server.

Slide 17

Let us understand HttpServletRequest interface.

HttpServletRequest Interface

- ❖ It defines an HttpServletRequest object, which is passed as an argument to the service() method.
- ❖ The methods defined by the HttpServletRequest interface are as follows:

public Cookie[] getCookies()
public String getHeader (String name)
public String getMethod()
public String getPathInfo()
public String getAuthType()



© Aptech Ltd. Web Component Development Using Java/Session 2 17

Use slide 17 to explain HttpServletRequest interface.

Tell them that the HttpServletRequest interface extends from the ServletRequest interface. It provides request information for HTTP servlets.

Explain each method defined by the HttpServletRequest interface:

- `public Cookie[] getCookies()` - Returns an array containing the entire Cookie object and returns null if no cookies were found.
For example, `Cookie[] cookies = request.getCookies();`
- `public String getHeader (String name)` - Returns the value of the specified request header as a String and returns null if the request did not include a header of the specified name.
For example, `out.println(request.getHeader("host") + "
");` returns the value of the header 'host'.
- `public String getMethod()` - Returns the name of the HTTP method used to make the request. For example, `out.println("<td align=center>" + request.getMethod() + "</td></tr>");`
- `public String getPathInfo()` - Returns the path information associated with a URL.
For example, `String ses_id1 = req.getPathInfo();`
- `public String getAuthType()` - Returns the basic authentication scheme used to protect the servlet from unauthorized users.
For example, `out.println(request.getAuthType());`

Slide 18

Let us understand reading parameters from request.

Reading Parameters from Request

- ❖ Request parameters:
 - Are strings sent by the client to a servlet container as a part of its request.
 - Are stored as a set of name-value pairs.
- ❖ Following methods of the `ServletRequest` interface are available to access parameters:
 - `public java.lang.String getParameter(java.lang.String name)`
 - Returns the value of a request parameter as a String, or null if the parameter does not exist.
 - `public java.util.Enumeration getParameterNames()`
 - Returns an enumeration of string objects containing the names of the parameters of this request.
 - `public java.lang.String[] getParameterValues(java.lang.String name)`
 - Returns an array of string objects containing all of the parameter values or null if parameters do not exist.

© Aptech Ltd. Web Component Development Using Java/Session 2 18

Use slide 18 to explain reading parameters from request.

Tell them request parameters sent by the client to a servlet container are in the form of strings. When the request is an `HttpServletRequest`, then the attribute values are populated from a query string. The data can also be sent through the request body, which is sent when the request method used is **POST**. The parameters are stored as a set of name-value pair by the servlet container.

Note, that for a given parameter name, there can be multiple parameter values. Some of the methods of the `ServletRequest` interface that can be used to access parameters within the servlet class are as follows:

- `public java.lang.String getParameter(java.lang.String name)` - For HTTP servlets, parameters are contained in the query string or posted form data. For example, `String name= request.getParameter("names");`
- `public java.util.Enumeration getParameterNames()` – If the request has no parameters, the method returns an empty enumeration. For example, `Enumeration Books = req.getParameterNames();`
- `public java.lang.String[] getParameterValues(java.lang.String name)`
 - Returns an array of String objects containing all the parameter values associated with a parameter name. For example, `String[] value = request.getParameterValues("interests");`

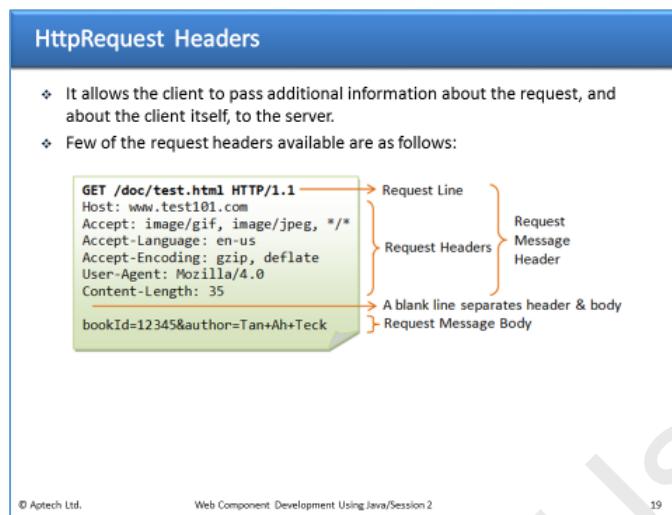
Tips:

All the form data sent either through query string or POST method are aggregated into the parameter set and then accessed using request parameter. However, the order of the aggregation makes the query string data to be sent before the post body data is sent.

For example, if a request data is sent using a query string, $x=30$, as well as post method body, $x=10 \& x=20$. The resulting parameter set would be ordered as: $x = (30, 10, 20)$.

Slide 19

Let us understand `HttpRequest Headers`.



Use slide 19 to explain `HttpRequest headers`.

Tell them that an HTTP client sends an HTTP request to a server in the form of a request message. The request message includes a Request Line, Request Message Headers, a blank line separating the header and body, and Request Message Body.

Then, explain some of the available request headers such as:

- `Accept` - It is used to specify types of headers acceptable by the client.
- `Accept-Charset` - The character sets acceptable by the response is specified by this field.
- `Accept-Encoding` - Restriction of content-coding which is accepted by response is mentioned in the field.
- `Accept-Language` - Restriction of a group of natural languages which is used by response is set by this field.
- `Authorization` - Authentication of a user agent with a server is done by this field.

Then, explain the figure given on slide 19 to identify the different parts of the request headers.

Slide 20

Let us understand reading headers from request.

Reading Headers form Request

❖ A servlet can access the headers of an HTTP request through the following methods of the `HttpServletRequest` interface.

```
public String getHeader (String name)
```

- Returns the value of the specified request header as a String.

```
public java.util.Enumeration getHeaders(java.lang.String name)
```

- Returns all the values of the specified request header as an enumeration of String objects.

```
public java.util.Enumeration getHeaderNames ()
```

- Returns an enumeration of all the header names this request contains.

© Aptech Ltd. Web Component Development Using Java/Session 2 20

Use slide 20 to explain reading headers from request.

Tell that a servlet can access the headers of an HTTP request through the following methods of the `HttpServletRequest` interface:

- `getHeader` method – This method returns a header, if the name of the header is given. However, it is possible that there can be multiple headers with the same name. For example, in an HTTP request, Cache-Control headers. If there are multiple headers with the same name, the `getHeader` method returns the first head in the request.
- `getHeaders` method - The `getHeaders` method allows access to all the header values associated with a particular header name, returning an Enumeration of String objects.

Ask them what if the header data contains `int` or `Date` data in the request. Then, how to access them? After listing to the answers, tell them that `HttpServletRequest` interface provides methods that allow access to header data in the mentioned format. The methods are as follows:

- `getIntHeader()`
- `getDateHeader()`

These methods translate the header value to the appropriate data type. In case, if the `getIntHeader()` method cannot translate the value to `int`, then a `NumberFormatException` is thrown. Similarly, the `getDateHeader()` method throws an `IllegalArgumentException`.

- `getHeaderNames` method - Returns an enumeration of all the header names this request contains. If the request has no headers, this method returns an empty enumeration. For example,

```
Enumeration strHeaderName = request.getHeaderNames();
```

Slides 21 to 23

Let us understand HTML forms.

HTML Forms 1-3

- ❖ HTML forms:
 - Are used to collect user input data.
 - Are stored as form fields and passed through browser URL for further processing in the Servlet.
- ❖ Several input types are provided by HTML to be used with `<form>` element. Some of them are as follows:

Text input

- Allows the user to enter a single line of text.

Drop-down list input

- Provides a list of options to the user to be selected.

Submit button

- Allows the user to send the data to the server for processing.

© Aptech Ltd. Web Component Development Using Java/Session 2 21

HTML Forms 2-3

- ❖ The HTML form has the following two major attributes:

```

graph LR
    Form[HTML Form] --> Method[Method]
    Form --> Action[Action]
    Method --> Get[Get]
    Method --> Post[Post]
    Action --> Includes[Includes the servlet class name]
  
```

© Aptech Ltd. Web Component Development Using Java/Session 2 22

HTML Forms 3-3

- ❖ The code snippet shows the HTML page for accepting the user choice.

```

<form method="GET" action="Login">
  <Username: <input type="text" name="username"/>
  <br/>

  <select name="color" size="1">
    <option> light
    <option> amber
    <option> brown
    <option> dark
  </select>

  <center>
    <input type="SUBMIT">
  </center>
</form>
  
```

© Aptech Ltd. Web Component Development Using Java/Session 2 23

Use slides 21 to 23 to explain HTML forms. HTML forms comprises input components such as text fields, radio buttons, checkboxes, clickable buttons, submit and reset buttons, and so on.

An HTML form is defined by the `<form>` tag.

Use the figure given on slide 22 to explain the major attributes, **Method** and **Action**, of HTML form element. Tell them that **method** attribute defines the way data transfer is done between the client and Web server. The methods such as Get, POST, HEAD, and so on are used to forward the parameters with the HTTP request.

- GET - With the GET method, all form data is encoded into the URL, appended to the action URL as query string parameters. The GET method is taken as the default if the method attribute is not specified in the HTML form.
- POST - With the POST method, form data appears within the message body of the HTTP request. It allows the client to send unlimited data in a single request. This method is useful if there is an important data such as credit card number to be sent to the Web server.
- HEAD – It is similar to GET method. However, the client sends a HEAD request when it wants to view only the headers of a response, such as Content-Type or Content-Length. The container must write the headers, before sending the response body. This method is commonly used for testing hyperlinks to obtain information related to their accessibility and modification.

Further explain that **action** attribute includes the servlet class name. For example, `<form method="POST" action="Login">`. Action attribute points to a file that acts as processing code for the form data, saving the data, composing an e-mail, and so on. The form will not be sent if the **action** attribute is not present.

Then, use the code snippet given on slide 23 to display the HTML page for accepting the user choice.

In-Class Question:

After you finish explaining HTML forms, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which tag defines an HTML form?

Answer:

The `<form>` tag.

Slide 24

Let us understand handling form data using request object.

Handling Form Data Using Request Object

- ❖ The request object:
 - Allows the servlet to obtain the user data from the form.
 - Calls the data reading methods of the request object.
- ❖ The code snippet shows the use of `getParameter()` method in handling data.

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws IOException,
ServletException
{
    String name = request.getParameter("username");
    String selectedColorValue =
request.getParameter("color");
    // data processing code here
}
```
- ❖ The code invokes the `getParameter()` method to get value of the fields `name` and `color` and stores them as `String`.

© Aptech Ltd. Web Component Development Using Java/Session 2 24

Use slide 24 to explain handling form data using request object.

Tell them that in Servlet, `doGET()` and `doPOST()` are the methods that handle form data. The `request` object allows the servlet to obtain the user data from the form.

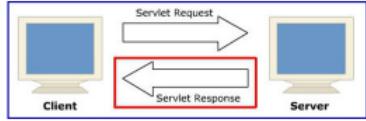
Use the code snippet given on slide 24 to display the use of `getParameter()` method in handling data. Tell them that the code invokes the `getParameter()` method to get value from the fields namely, `name` and `color` and stores them as string.

Slides 25 and 26

Let us understand `ServletResponse` interface.

ServletResponse Interface 1-2

- ❖ The `ServletResponse` and `HttpServletResponse` interfaces include all the methods needed to create and manipulate a servlet's output.
- ❖ Figure depicts servlet response.



- ❖ The `ServletResponse` interface:
 - Defines methods that allow to retrieve an output stream to send data to the client, decide on the content type, and so on.
 - Is passed as an argument to `service()` method of a servlet.

© Aptech Ltd. Web Component Development Using Java/Session 2 25

ServletResponse Interface 2-2

- ❖ The methods defined by the `ServletResponse` interface are as follows:

```
public java.lang.String getContentType()
public PrintWriter getWriter() throws IOException
public ServletOutputStream getOutputStream() throws IOException
public void setContentType(java.lang.String str)
```

© Aptech Ltd. Web Component Development Using Java/Session 2 26

Use slides 25 and 26 to explain `ServletResponse` interface.

Tell them that handling `ServletResponse` object means to help a servlet in sending a response to the client using different interfaces.

Use the figure given on slide 25 to depict the servlet response. Tell them that the `ServletResponse` interface defines an object, which is used to provide responses to the clients. The `ServletResponse` object is passed as an argument to `service()` method of a servlet by the servlet container.

Use slide 25 to explain the methods defined by the `ServletResponse` interface with examples as:

- `public java.lang.String getContentType()` - Returns the Multipurpose Internet Mail Extensions (MIME) type of the request body or null if the type is not known. For example,
`out.println ("Message contenttype:"+ msg.getContentType () + "
");`
- `public PrintWriter getWriter() throws IOException` - Returns an object of `PrintWriter` class that sends character text to the client. For example, `PrintWriter out1= response.getWriter(); out1.println("Sometext and HTML");`

- `public ServletOutputStream getOutputStream() throws IOException` - Uses `ServletOutputStream` object to write response as binary data to the client. For example,
`ServletOutputStream out2 = response.getOutputStream(); out2.write(twoBytearray);`
Note that the servlet container does not encode the binary data.
- `public void setContentType(java.lang.String str)` - Used to set the format in which the data is sent to the client, either in normal text format or html format. For example,
`response.setContentType("text/html");`

Slide 27

Let us understand `HttpServletResponse` Interface.

HttpServletResponse Interface

- ❖ The `HttpServletResponse` interface extends `ServletResponse` interface and provides information to an `HttpServlet`.
- ❖ It defines an `HttpServlet` object, which is passed as an argument to the `service()` method of a servlet.
- ❖ Figure depicts HTTP servlet response.

```
graph TD; WebClient[Web Client] -- "HTTP Request" --> ServletContainer[Servlet Container]; subgraph ServletContainer [Servlet Container]; direction TB; subgraph SL [Servlet Logic]; doGetPost["doGet/doPost method"]; end; SL <--> SR[Servlet Response]; SR -- "HTTP Response" --> WebClient;
```

© Aptech Ltd. Web Component Development Using Java/Session 2 27

Use slide 27 to explain `HttpServletResponse` interface.

Tell the students that using the `HttpServletResponse` interface, you can set HTTP response header, set the Content-Type of the response, acquire a text stream for the response, acquire a binary stream for the response, and redirect an HTTP request to another URL or add cookies to the response.

Use the figure given on slide 27 to explain HTTP servlet response.

Slides 28 and 29

Let us understand methods in `HttpServletResponse` interface.

Methods in `HttpServletResponse` Interface 1-2

- The methods defined by the `HttpServletResponse` interface are as follows:

`public void addCookie(Cookie cookie)`

- Adds the specified cookie to the response, sent to the client.
- The code snippet shows the use of `addCookie()` method.

```
/* Adds a cookie named cookie2 with color red to
the existing cookies */
Cookie cookie2 = new Cookie("color", "red");
response.addCookie(cookie2);
```

© Aptech Ltd. Web Component Development Using Java/Session 2 28

Methods in `HttpServletResponse` Interface 2-2

- `public void addHeader(java.lang.String name, java.lang.String value)`
 - Used to add name and value to the response header.
- `public boolean containsHeader(String name)`
 - Used to verify if the response header contains any values.
 - It returns true if the response header has any values.
 - It returns false otherwise.
- `public boolean sendError(int sc) throws java.io.IOException`
 - Sends an error response to the client using the specified status code and clearing the buffer.
 - Status codes are used to indicate the reason, a request is not satisfied or that a request has been redirected.

© Aptech Ltd. Web Component Development Using Java/Session 2 29

Use slides 28 and 29 to explain the methods defined by the `HttpServletResponse` interface.

Explain the methods provided by the `HttpServletResponse` interface:

- `public void addCookie(Cookie cookie)` - Adds the specified cookie to the response sent to the client.
- `public void addHeader(java.lang.String name, java.lang.String value)` – Example: `response.addHeader("Refresh", "15")`; Tell them that if there is no header associated with the name, then a new header is created.
- `public boolean containsHeader(String name)` -
`res.containsHeader("Cache")`;
- `public void sendError(int sc) throws java.io.IOException`-
`httpResp.sendError(HttpServletResponse.SC_FORBIDDEN, "Good Bye for now!")`; The `SendError()` method will set the appropriate headers and content body for an error message to be returned back to the client. It sends the error to the client using the specified status code.

Slide 30

Let us understand response headers.

Response Headers

- ❖ Figure shows the parameters of response header sent in response to the client.

```
$ curl -I searchengineland.com
HTTP/1.1 200 OK
Date: Sat, 06 Aug 2011 03:57:00 GMT
Server: Apache
Last-Modified: Sat, 06 Aug 2011 00:09:14 GMT
ETag: "374c333-13de3-4a9cb062e5e80"
Accept-Ranges: none
Content-Length: 81379
Cache-Control: max-age=300, must-revalidate
Expires: Sat, 06 Aug 2011 04:02:00 GMT
Vary: Accept-Encoding,Cookie
Content-Type: text/html; charset=UTF-8
```

© Aptech Ltd. Web Component Development Using Java/Session 2 30

Use slide 30 to explain response headers.

Tell them that the response header is attached to the files being sent back to the client. It contains the date, size, and type of file that the server sends back to the client and also data about the server itself. Response headers can be used to specify cookies to supply the modification date. It is also used to instruct the browser to reload the page. In addition, it specifies how big the file is, to determine how long the HTTP connection needs to be maintained.

Use the figure given on slide 30 to display the parameters of response header sent in response to the client.

Slide 31

Let us understand Sending Headers.

Sending Headers

❖ The methods of `HttpServletResponse` interface that are used to send header information to the client are as follows:

- `addHeader()`
- `addDateHeader()`
- `addIntHeader()`
- `containsHeader()`

© Aptech Ltd. Web Component Development Using Java/Session 2 31

Use slide 31 to explain sending headers.

Explain each method with an example as shown:

- `addHeader()` - Adds a response header with the given name and value.
For example, `response.addHeader("Cache", 3.14);`
- `addDateHeader()` - Adds a response header with the given name and date value.
For example, `res.addDateHeader("Cache", 20-02-2002)`
- `addIntHeader()` - Adds a response header with the given name and integer-value.
For example, `res.addIntHeader("Cache", 3)`
- `containsHeader()` - Returns a boolean value to indicate if response header has already been set.

Slide 32

Let us understand sending data from servlet using response object.

Sending Data from Servlet Using Response Object

- ❖ The response object should perform the following steps:
 - Inform the browser, the type of file/data that is getting transferred.
 - Convert the object data to stream.
- ❖ The code snippet shows how to send data using response object.

```
public class CodeReturnJARfile extends HttpServlet {  
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {  
        response.setContentType("application/jar");  
        ServletContext ctx = getServletContext();  
        InputStream is = ctx.getResourceAsStream("/JAVAcompactV3.jar");  
        int read = 0;  
        byte[] bytes = new byte[1024];  
        OutputStream os = response.getOutputStream();  
  
        while ((read = is.read(bytes)) != -1) {  
            os.write(bytes, 0, read);  
        }  
        os.flush();  
        os.close(); } }
```

© Aptech Ltd. Web Component Development Using Java/Session 2 32

Use slide 32 to explain creating a servlet to send the jar file data using response object to the client.

Use the code snippet given on slide 32 to demonstrate how to send data using the response object.

Tell them that, in the code snippet, `response.setContentType("application/jar")`; tells the browser to recognize that the response object carries a JAR.

The statement, `OutputStream os= response.getOutputStream()`; reads the JAR bytes and writes the bytes to the output stream that is received from the response object.

Slide 33

Let us understand reading binary data.

Reading Binary Data

- ❖ Multipart/form-data is an Internet media type that returns a set of values as the result of a user filling out a form.
- ❖ These set of values are transmitted as binary data.
- ❖ Each request handled by a servlet has an input stream associated with it.
- ❖ Use `getInputStream()` to retrieve the input stream as a `ServletInputStream` object.
- ❖ **Syntax:**

```
public ServletInputStream  
ServletRequest.getInputStream() throws  
IOException
```

© Aptech Ltd. Web Component Development Using Java/Session 2 33

Use slide 33 to explain reading of binary data.

Tell them that some Web applications such as e-mail need to send files such as documents and images from one system to another as attachment. In this case, data transmitted from a client to a Web application will be of binary type. The user will specify the reference to the binary files in online form.

Multipart/form-data is an Internet media type that returns a set of values as the result of a user filling out a form. These set of values are transmitted as binary data. Each request handled by a servlet has an input stream associated with it.

Then, explain the syntax of the method `getInputStream()` as displayed on slide 33. It is used to retrieve the input stream as a `ServletInputStream` object.

Slides 34 and 35

Let us understand producing text and binary data.

Producing Text and Binary Data

- ❖ The methods belonging to `ServletResponse` interface for producing output streams are as follows:
 -  `public ServletOutputStream getOutputStream() throws java.io.IOException`
 - The `getOutputStream()` returns a `ServletOutputStream`, which can be used for binary data.
 -  `public java.io.PrintWriter getWriter() throws java.io.IOException`
 - The `getWriter()` returns a `java.io.PrintWriter` object, which is used only for textual output.
 - The `getWriter()` method examines the content-type to determine what character set to use.

© Aptech Ltd. Web Component Development Using Java/Session 2 34

Sending Text and Binary Data

- ❖ `ServletOutputStream` provides an output stream for sending binary data to the client.
- ❖ The methods supported by the `ServletOutputStream` class are as follows:
 - `public void print(boolean b) throws java.io.IOException`
 - Writes a boolean value to the client with no Carriage Return-line Feed (CRLF) character at the end.
 - `public void println(char c) throws java.io.IOException`
 - Writes a character value to the client, followed by a Carriage Return-line Feed (CRLF).

© Aptech Ltd. Web Component Development Using Java/Session 2 35

Use slide 34 to explain producing text and binary data.

Two methods belonging to `ServletResponse` interface for producing output streams are as follows:

- `public ServletOutputStream getOutputStream() throws java.io.IOException`
- `public java.io.PrintWriter getWriter() throws java.io.IOException`

The `getWriter()` returns a `java.io.PrintWriter` object, which is used only for textual output. The `getWriter()` method examines the content-type to determine what character set to use, so `setContentType()` should be called, before `getWriter()`.

Use slide 35 to explain sending text and binary data. Tell them that whenever any file sent to the client has to be encoded, for example, images or text attachments, it is sent as binary data.

Slide 36

Let us understand redirecting requests.

Redirecting Requests

- ❖ Requests are redirected by following methods:

```
public void sendRedirect(java.lang.String location) throws
java.io.IOException
```

- Sends a redirect response to the client using the specified redirect location URL.

```
public java.lang.String encodeRedirectURL(java.lang.String url)
```

- Encodes the specified URL for use in the `sendRedirect()` method, or if encoding is not needed, returns the URL unchanged.
- All URLs sent with the `sendRedirect()` method should be run encoded using this method.

© Aptech Ltd. Web Component Development Using Java/Session 2 36

Use slide 36 to explain redirecting requests.

Tell the students that whenever the client makes a request, the request goes to the servlet container. The Servlet container decides whether the concerned servlet can handle the request or not. If the servlet cannot handle the request, it decides that the request can be handled by another servlet or JSP. Then, the servlet calls the `sendRedirect()` method and sends the response to the browser along with the status code. The browser makes a temporary new request, with the name of the servlet that can now handle the request and displays the result on the browser.

Provide an example for `public void sendRedirect(java.lang.String location) throws java.io.IOException` method as:

```
response.sendRedirect(response.encodeRedirectURL(contextPath +
"/maps"));
```

Tips:

The `sendRedirect()` method sends a temporary redirect response to the client using the specified redirect location URL. This method can accept relative URLs; the servlet container must convert the relative URL to an absolute URL before sending the response to the client. If the location is specified without a leading '/', then the container interprets it as relative to the current request URI. However, if the location is specified with a leading '/', then the container interprets it as relative to the servlet container root.

Slides 37 to 40

Let us understand request dispatcher.

Request Dispatcher 1-4

- ❖ A servlet runs in an environment called the servlet context, which describes various parameters associated with the servlet.
- ❖ A servlet belongs to only one servlet context.
- ❖ A servlet cannot access resources such as static HTML pages, which are stored in local files using the RequestDispatcher objects.
- ❖ The local resource can be accessed using the method `getResource(String path)` of `ServletContext` object that returns a `URL` object for a resource specified by a local Uniform Resource Identifier (URI), for example, '/html1'.
- ❖ The method `ServletRequest.getRequestDispatcher(java.lang.String)` allows using relative path, whereas the method `ServletContext.getRequestDispatcher(java.lang.String)` allows only absolute path.

Request Dispatcher 2-4

- ❖ The servlets belonging to same application can share data using the following methods of `RequestDispatcher` interface are as follows:

forward() method

- ❑ The method is used to forward request from one servlet to another resource on the same server.
 - ❑ **Syntax:**
- ```
public void forward(ServletRequest request1,
 ServletResponse response1) throws
 ServletException, IOException
```
- where,
- `Request1` – is the request made by the client to the servlet.
  - `response1` – is the response made by the servlet to the client.

### Request Dispatcher 3-4

- The code snippet retrieves the servlet context from its RequestDispatcher instance, and then forwards control to the JSP page.

```
public class MyServlet extends HttpServlet
{
 public void doGet(HttpServletRequest request,
 HttpServletResponse response) throws ServletException,
 IOException {
 request.setAttribute("title", "Home Page");
 ServletContext servletContext = getServletContext();
 RequestDispatcher dispatcher = servletContext.
 getRequestDispatcher("/cart.jsp");
 dispatcher.forward(request, response);
 }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 2

39

### Request Dispatcher 4-4

#### include() method

- The method is used to include the contents of another servlet, JSP page, or an HTML file.
- Syntax:**

```
public void include(ServletRequest request1, ServletResponse
 response) throws ServletException, IOException
 void println(char c) throws java.io.IOException
```

- The code snippet demonstrates how to include the content from the specified path.

```
// Include static page or servlet/JSP page from the specified path,
...
RequestDispatcher dispatcher = getServletContext().
getgetRequestDispatcher("/contactus.jsp");
if (dispatcher == null)
 out.println(path + " not found");
else
 dispatcher.include(request, response);
...
```

© Aptech Ltd.

Web Component Development Using Java/Session 2

40

Use slides 37 to 40 to explain request dispatcher. Tell them that the servlets sometimes need to access network resources to satisfy client requests. The resources can be another servlet, a JSP page, or a CGI script. In other words, we need to dispatch the request from one component to another component. This can be done by using RequestDispatcher interface.

Tell them that RequestDispatcher interface is implemented by servlet container to dispatch or to pass the request to a Web resource such as Servlet, HTML page, or JSP page.

To obtain the RequestDispatcher object, you can use the following methods:

- The getRequestDispatcher () method of ServletContext
- The getRequestDispatcher () method of HttpServletRequest

#### getRequestDispatcher () method of ServletContext

This method takes **String** argument to locate the resource to which request is to be dispatched. When the method is called, the container locates given path which should start with the '/' character. If given path does not start with '/' character it throws **IllegalArgumentException**.

For example, `requestDispatcher reqDispObj =  
getServletContext().getRequestDispatcher("/ContextRoot/home.jsp");`

### **getRequestDispatcher () method of ServletRequest**

This method obtains the RequestDispatcher object using path to the current request. The Servlet container builds complete path and locates the resource provided in the getRequestDispatcher() method of ServletContext. For example,

```
requestDispatcher reqDispObj =
getServletContext().getRequestDispatcher("/home.jsp");
```

Then, tell them that to dispatch the request using RequestDispatcher, perform the following steps:

- Get a RequestDispatcher object reference
- Using include() and forward() methods of RequestDispatcher

### **Using forward() method**

This method checks whether servlet has obtained the response and output in the response buffer. If buffer is not committed, the content is cleared. However, if the buffer is already committed, it throws IllegalStateException. It implies that after invoking forward() method, the Servlet cannot add any response content.

Then, using the syntax provided on slide 38 to display and explain the syntax of forward() method. Use the code snippet given on slide 39 to explain how to retrieve the RequestDispatcher instance and then forward the control to the JSP page.

### **Using include() method**

This method includes the response of another Servlet into the calling Servlet. This method can be invoked from calling Servlet while servicing the request. It includes contents of resource such as Servlet, JSP page, or HTML page in the response. If we want generate response in the source servlet, then we should make use of include () method.

Using slide 40, explain the syntax of include () method. Then, explain the code snippet as shown on slide 40 to demonstrate the code that includes the content from the specified path.

## Slides 41 and 42

Let us understand init parameters.

**init Parameters 1-2**

- ❖ The `ServletConfig` interface provides various methods to configure a servlet, before processing the data requested by the client.
- ❖ The servlet has the direct access to Servlet initialization parameters using the `getInitParameter()` method.
- ❖ **Syntax of `getInitParameter()` method:**

```
public String getInitParameter(String name)
```

- ❖ The code snippet demonstrates how to retrieve the parameters of the servlet.

```
//Retrieves the parameters user and password
super.init(config);
String userl = getInitParameter("user");
String passwordl = getInitParameter("password");
...
...
```

© Aptech Ltd. Web Component Development Using Java/Session 2 41

**init Parameters 2-2**

- ❖ Figure depicts servlet initialization.

```
18 public class InitServlet extends HttpServlet {
19 protected void doPost(HttpServletRequest request,
20 HttpServletResponse response)
21 throws ServletException, IOException {
22 response.setContentType("text/html;charset=UTF-8");
23 PrintWriter out = response.getWriter();
24 /* TODO output your page here */
25 out.println("<html>");
26 out.println("<head>");
27 out.println("<title>"+InitServlet.title+"</title>");
28 out.println("</head>");
29 out.println("<body>");
30 out.println("InitServlet at "+request.getContextPath()
31 () + "</body>");
32 String initParam=super.getServletConfig().getInitParameter("MyParam");
33 out.println("Value of the init parameter is :"+initParam);
34 out.println("</body>");
35 out.println("</html>");
36 }
out.close();
```

© Aptech Ltd. Web Component Development Using Java/Session 2 42

Use slides 41 and 42 to explain init () parameters.

Tell them that in a Web application, the database connection information, such as SQL username and password are not sent to the servlet every time a client makes a request. Hence, this information needs to be passed to the servlet before beginning the servlet life cycle.

To pass as an argument during initialization, the servlet container uses an object of `ServletConfig` interface. A servlet configuration object is used by a servlet container to pass information to a servlet during initialization.

Using slide 41, explain the syntax of using the `getInitParameter()` method. This method retrieves the value of the initialization parameter and returns null if the specified parameter does not exist. Then, use the code snippet as shown on slide 41 to demonstrate how to retrieve the parameters of the servlet. Tell them that the same method is available in the `ServletConfig` interface to read the initialization parameters. Then, use the figure given on slide 42 to explain servlet initialization.

### Tips:

To read all the initialization parameters within the servlet, the `getInitParameterNames()` from the `ServletConfig` interface can be invoked. The method returns the enumeration of string objects.

### Slide 43

Let us understand reading parameters in the `init()` method.

**Reading Parameters in init() Method**

```
public void init() throws ServletException
```

◆ The code snippet illustrates how a servlet reads initialization parameters.

```
int count;
public void init(ServletConfig config) throws
ServletException {
super.init(config);
String initial = config.getInitParameter("initial");
try {
count = Integer.parseInt(initial);
} catch (NumberFormatException e) {
count = 0;
}
}
```

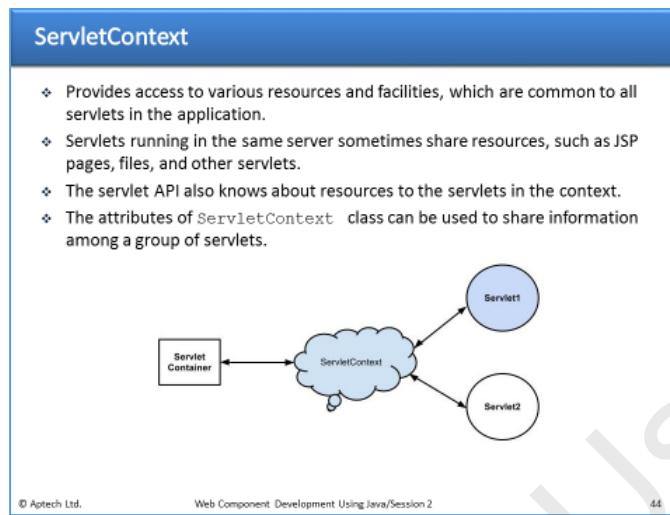
© Aptech Ltd.      Web Component Development Using Java/Session 2      43

Use slide 43 to explain reading parameters in the `init()` method.

Use the code snippet given on slide 43 to illustrate how a servlet reads initialization parameters within the `init()` method.

## Slide 44

Let us understand ServletContext.



Use slide 44 to explain `ServletContext` interface. Tell them that the container first locates the servlet class, loads the servlet, and creates an instance of the servlet. It then invokes `init()` method to initialize the servlet. A `ServletConfig` interface object is passed as an argument to the `init()` method, which provides the servlet with access to `ServletContext` interface.

The servlet context provides access to various resources and facilities, which are common to all servlets in the application. The servlet API is used to set the information common to all servlets in an application. Servlets running in the same server sometimes share resources, such as JSP pages, files, and other servlets.

This is required when several servlets are bound together to form a single application, such as a chat application, which creates single chat room for all users. The servlet API also knows about resources to the servlets in the context. The attributes of `ServletContext` class can be used to share information among a group of servlets.

## Slides 45 to 47

Let us understand ServletContext methods.

### ServletContext Methods 1-3

- ❖ The `ServletContext` methods are used to get and set additional information about the servlet.
  - ❖ Some commonly used methods of `ServletContext` interface are as follows:
- ```
public String getInitParameter(String name)
public void setAttribute(String name, Object object)
public Object getAttribute(String name)
public Enumeration getInitParameterNames()
public void removeAttribute(String name)
```

ServletContext Methods 2-3

- ❖ The code snippet shows the `web.xml` file for specifying the context parameters and servlet initialization parameters.

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
. . .
<context-param>
    <param-name>Global</param-name>
    <param-value>ValueToAll</param-value>
</context-param>
<!-- other stuff including servlet declarations -->
<servlet>
    <servlet-name>ParamServletTests</servlet-name>
    <servlet-class>TestInitParams</servlet-class>
<init-param>
    <param-name>ServletSpecific</param-name>
    <param-value>ServletValue</param-value>
</init-param>
</servlet>
</web-app>
```

ServletContext Methods 3-3

- ❖ The code snippet demonstrates how to read the context parameters from the `web.xml` file.

```
. . .
public class ContextParameters extends HttpServlet{
    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter pw = res.getWriter();
        // Creating ServletContext object
        ServletContext context=getServletContext();
        //Getting the value of the initialization parameter and
        //printing it
        String str = context.getInitParameter("Global");
        pw.println("driver name is="+ str);
        pw.close();
    }
}
```

Use slides 45 to 47 to explain ServletContext methods.

Explain some commonly used methods of `ServletContext` interface in slide 45 as:

- `public String getInitParameter(String name)` - returns the parameter value for the specified parameter name.
- `public void setAttribute(String name, Object object)` - sets the given object in the application scope.
- `public Object getAttribute(String name)` - returns the attribute for the specified name.
- `public Enumeration<String> getInitParameterNames()` - returns the names of the context's initialization parameters as an Enumeration of String objects.
- `public void removeAttribute(String name)` - removes the attribute with the given name from the servlet context.

Use the code snippet given on slide 46 to display the `web.xml` file for specifying the context parameters and servlet initialization parameters.

Use the code snippet given on slide 47 to demonstrate the code that reads the context parameters from the `web.xml` file.

Slides 48 and 49

Let us understand status codes.

Status Codes 1-2

- ❖ Table lists some of the error codes along with their associated message and meaning.

Status Code	Associated Message	Meaning
301	Moved Permanently	Document is moved to a separate location as mentioned in the URL. The page is redirected to the mentioned URL.
302	Found	Temporary replacement of file from one location to the other as specified.
400	Bad Request	The request placed is syntactically incorrect.
401	Unauthorized	Authorization is not given to access a password protected page.
404	Not Found	Resource not found in the specified address.
408	Request Timeout	Time taken by client is very long to send the request (only available in HTTP 1.1).
500	Internal Server Error	Server is unable to locate the requested file. The servlet has been deleted or crashed or has been moved to a new location without informing.

© Aptech Ltd.

Web Component Development Using Java/Session 2

48

Status Codes 2-2

- ❖ Figure depicts an example of status code.

© Aptech Ltd.

Web Component Development Using Java/Session 2

49

Use slides 48 and 49 to explain error handling in Servlets and their status codes.

Tell them that there are many situations where a Web server can give an error. A requested page may be moved from one location to another. The address may be wrongly typed. The requested page may be forbidden, may be temporarily deleted, or correct HTTP version might not have found. There are other situations where an error may be generated.

Errors may occur due to one reason or the other, but the status code for each type of error, makes it easy for identifying the error. Use the table given on slide 48 to explain the list of some of the error codes along with their associated message and meaning.

Use the figure given on slide 49 to depict an example of status code.

Slides 50 and 51

Let us understand error methods in ‘`HTTPServlet`’ class.

Error Methods in ‘`HttpServlet`’ Class 1-2

- ❖ Errors during the execution of a Web application are reported using the following methods:

`sendError()`

- ❑ This method checks for the status code and sends to the user the specified response message.
- ❑ After sending the error message, the buffer is cleared.
- ❑ Syntax:

```
public void sendError(int sc1) throws
java.io.IOException
```

- ❑ The code snippet shows the use of `sendError()` method.

```
// Return the file file1
try {
    ServletUtils.returnFile(file1, out);
} catch (FileNotFoundException err) {
    res.sendError(res.SC_NOT_FOUND);
}
```

© Aptech Ltd. Web Component Development Using Java/Session 2 50

Error Methods in ‘`HttpServlet`’ Class 2-2

`setStatus()`

- ❑ This code is specified earlier so that on receiving the `setStatus()` method, the error message is thrown or redirected to another default Web page.
- ❑ Syntax:

```
public void HttpServletResponse.setStatus(int sc1)
```

- ❑ The code snippet shows the use of `setStatus()` method.

```
/* If the client sent an If-Modified-Since header equal or
after the servlet's last modified time, send a short "Not
Modified" status code. Round down to the nearest second since
client headers are in seconds */
if (servletLastMod == -1) {
    super.service(req, res);
} else if ((servletLastMod / 1000 + 1000) <=
req.getDateHeader("If-Modified-Since")) {
    res.setStatus(res.SC_NOT_MODIFIED);
}
```

© Aptech Ltd. Web Component Development Using Java/Session 2 51

Use slides 50 and 51 to explain the methods that are used to report errors during the execution of a Web application.

Using slide 50, explain the syntax of `sendError()` method. Tell them that in the syntax, `sc1` is a status code passed as an integer value to the method and the `sendError()` method sends a predefined error message as a response for the ‘file not found’.

Use the code snippet given on slide 50 to display the use of `sendError()` method.

Using slide 51, explain the syntax of `setStatus()` method. Tell them that in the syntax, `sc1` is the status code. Use the code snippet given on slide 51 to display the use of `setStatus()` method.

Slides 52 to 54

Let us understand logging errors.

Logging Errors 1-3

- ❖ Servlets can store the actions and errors through the `log()` method of the `GenericServlet` class.
 - ❖ The `log()` method also assists in debugging by describing all the details about an error.
 - ❖ The `log()` method can be used by passing either a single argument or two arguments.
 - ❖ **Syntax:**
- ```
public void log(String msg)
```
- ❖ The error can be viewed by using `log()` to record in a server.

© Aptech Ltd.

Web Component Development Using Java/Session 2

52

### Logging Errors 2-3

- ❖ The code snippet shows the use of `log()` methods.
- ```
// Return the file
try {
    ServletUtils.returnFile(file, out);
} catch (FileNotFoundException e) {
    log("Could not find file: " + e.getMessage());
    res.sendError(res.SC_NOT_FOUND);
} catch (IOException e) {
    log("Problem sending file", e);
    res.sendError(res.SC_INTERNAL_SERVER_ERROR);
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 2

53

Logging Errors 3-3

- ❖ Figure depicts a log file.

```
localhost_2007-05-19.log - Notepad
File Edit Format Help
[May 19, 2007 1:45:03 PM org.apache.catalina.core.ApplicationContext log
INFO: ContextListener: contextInitialized()]
[May 19, 2007 1:45:03 PM org.apache.catalina.core.ApplicationContext log
INFO: SessionListener: contextInitialized()]
[May 19, 2007 1:46:14 PM org.apache.catalina.core.ApplicationContext log
INFO: SessionListener: contextDestroyed()]
[May 19, 2007 1:46:14 PM org.apache.catalina.core.ApplicationContext log
INFO: ContextListener: contextDestroyed()]
[May 19, 2007 1:48:58 PM org.apache.catalina.core.ApplicationContext log
INFO: ContextListener: contextInitialized()]
[May 19, 2007 1:48:58 PM org.apache.catalina.core.ApplicationContext log
INFO: SessionListener: contextInitialized()]
[May 19, 2007 5:27:47 PM org.apache.catalina.core.ApplicationContext log
INFO: SessionListener: contextDestroyed()]
[May 19, 2007 5:27:47 PM org.apache.catalina.core.ApplicationContext log
INFO: ContextListener: contextDestroyed()]
```

© Aptech Ltd.

Web Component Development Using Java/Session 2

54

Use slides 52 to 54 to explain logging errors. Tell them that servlets can store the actions and errors through the `log()` method of the `GenericServlet` class. The message along with the stack's trace is thrown to a servlet log.

Using slide 52, explain the syntax of `log()` method. Mention that in the syntax, `msg1` is the specified message to be written to the servlet log file.

Use the code snippet given on slide 53 to display the use of `log()` methods. Tell them that the method writes an explanatory error message and a stack trace for specified `throwable` object into the servlet log file.

Use the figure given on slide 54 to depict a log file.

Slides 55 and 56

Let us understand forwarding to error page.

Forwarding to Error Page 1-2

- ❖ Following methods create object for the `RequestDispatcher` interface:

```
'ServletContext.getRequestDispatcher(String path)'
```

- This method takes a string argument as a path, which is within the scope of the `ServletContext`.
- The path, relative to root is passed as an argument.
- This path, in turn, is used to locate the servlet to which it is redirected.

```
'ServletContext.getNamedDispatcher(String name)'
```

- The method takes a string argument indicating the name of servlet to the `ServletContext`.
- It returns a `RequestDispatcher` object for the named servlet.

```
'ServletRequest.getRequestDispatcher(String path)'
```

- This method uses relative path which is the location of the file relative to current path.

© Aptech Ltd. Web Component Development Using Java/Session 2 55

Forwarding to Error Page 2-2

- ❖ The code snippet shows the dispatching of error object.

```
public class Dispatcher extends HttpServlet {
    public void doGet(HttpServletRequest req,
                      HttpServletResponse res) {
        RequestDispatcher dispatcher1 =
            request.getRequestDispatcher("/error_page.jsp");
        if (dispatcher1 != null)
            dispatcher1.forward(req, res);
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 2 56

Use slides 55 and 56 to explain forwarding to error page.

Tell them that a `RequestDispatcher` object is used to redirect the client request to a Web page on receiving an error message on the server. The `RequestDispatcher` object is created by the servlet container.

Using slide 55, explain the different methods to create the `RequestDispatcher` object and pass the path for the error page.

Use the code snippet given on slide 56 to display how to dispatch error object. Tell them that in the code, the `Dispatcher` class request for '`error_page.jsp`' file and wait for the response. If the `dispatcher1` is not null, then the file is forwarded as response to the request received from the `dispatcher` class.

©Aptech Limited

Slide 57

Let us summarize the session.

Summary

- ❖ The init(), service(), and destroy() methods are the servlet's lifecycle methods. The init() method intimates the servlet that it is being placed into service.
- ❖ The service() method is called by the servlet container to allow the servlet to respond to a request. The servlet container calls the destroy() method after all threads within the servlet's service method have exited or after a timeout period has passed.
- ❖ A GenericServlet class defines a servlet that is not protocol dependent. To have better control over the required servlets HttpServlet is extended to GenericServlet.
- ❖ A servlet request contains the data to be passed between a client and the servlet. All requests implement the ServletRequest interface, which defines the methods for accessing the relevant information.
- ❖ A servlet response contains data to be passed between a server and the client. All responses implement the ServletResponse interface. This interface defines various methods to process response.
- ❖ You use getInputStream() to retrieve the input stream as a ServletInputStream object. ServletOutputStream provides an output stream for sending binary data to the client.
- ❖ Resources are included to a servlet by forwarding request from one servlet to another by using the forward() and include() methods along with RequestDispatcher interface. Inter-servlet communication can be used by the servlets to gain access to other currently loaded servlets and perform some tasks on other servlets.
- ❖ Errors may occur due to one reason or the other but the status code for each type of error, makes it easy for identifying the error. Errors are reported using sendError() and setStatus() methods. Errors are logged using the log() method of ServletContext and forwarded to an error page using the RequestDispatcher interface.

© Aptech Ltd. Web Component Development Using Java/Session 2 57

In slide 57, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

2.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore various session management techniques that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 3 – Session Tracking

3.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

3.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe the stateless nature of HTTP protocol
- Explain the need of tracking client identity and state
- Explain the URL rewriting method for session tracking
- Explain how to use hidden form fields
- Explain the use of Cookie class and its methods
- Explain how to store and retrieve information in a session
- Describe the use of HttpSession interface and its methods
- Explain how to invalidate a session

3.1.2 Teaching Skills

To teach this session successfully, you should be aware yourself with the concept of tracking clients identity on the Web server. Familiarize yourself with various session techniques such as URL rewriting, hidden fields, cookies, and HttpSession API supported in Java Web applications, and the various methods used in these techniques.

You need to aware yourself with how to store and retrieve information in a session and also invalidating a session.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❖ Describe the stateless nature of HTTP protocol
- ❖ Explain the need of tracking client identity and state
- ❖ Explain the URL rewriting method for session tracking
- ❖ Explain how to use hidden form fields
- ❖ Explain the use of Cookie class and its methods
- ❖ Explain how to store and retrieve information in a session
- ❖ Describe the use of HTTP session interface and its methods
- ❖ Explain how to invalidate a session

© Aptech Limited. Web Component Development Using Java/Session 3 2

Tell the students that they will be introduced to session tracking. This session explains the concept of session tracking that allows the server to keep a track of successive requests made by the same client. The session explains various session tracking techniques, such as URL rewriting, hidden fields, cookies, and Session API that are used to maintain the client's information.

They will learn how to use `Cookie` class and its different methods. The session discusses on how to store and retrieve information from a user's session. Then, the session explains `HttpSession` interface and the different methods used in maintaining the client details in a session. Finally, it discusses about invalidating a session.

3.2 In-Class Explanations

Slides 3 to 5

Let us understand protocol and its features.

Introduction 1-3

- ❖ **Protocol**
 - A set of rules, which governs the syntax, semantics, and synchronization of communication in computers.

- ❖ A protocol is said to be stateless when:
 - The configuration setting, transaction, and information are not tracked by a protocol.
 - The connections last for only one transaction.

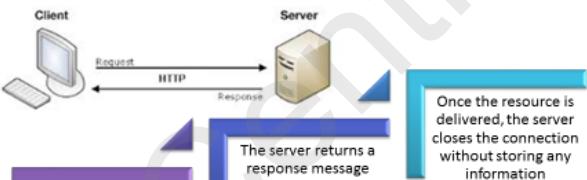
Example: HTTP protocol.



© Aptech Limited. Web Component Development Using Java/Session 3 3

Introduction 2-3

❖ Why HTTP is referred to as a Stateless Protocol?



The server returns a response message with the requested resource

Once the resource is delivered, the server closes the connection without storing any information

HTTP client opens a connection and sends a request message to an HTTP server for a resource

No connection information is stored, and hence **HTTP is referred to as a stateless protocol.**

© Aptech Limited. Web Component Development Using Java/Session 3 4

Introduction 3-3

❖ Advantages and Disadvantages of a Stateless Protocol

Advantages	Disadvantages
<ul style="list-style-type: none"> • Hosts do not need to retain information about users between requests. • Simplifies the server design. 	<ul style="list-style-type: none"> • Need to include more information in each request which would be interpreted by the server each time. • No acknowledgement of received information.

© Aptech Limited. Web Component Development Using Java/Session 3 5

Using slides 3 to 5, explain protocols and its features.

Tell the students that a protocol is typically a set of rules, which governs the syntax, semantics, and synchronization of communication.

Tell them the rules are used for the following purposes:

- a. For compressing the data.
- b. For sending device to indicate that it has finished sending a message.
- c. For receiving device to indicate that it has received a message.

Then, mention that HTTP is a stateless protocol that treats each request as an independent transaction that is unrelated to any previous request.

Mention that the best example for stateless protocol includes Internet Protocol (IP) and HTTP protocol. These protocols laid the foundation of data communication for the World Wide Web.

Working of HTTP Stateless Protocol

In the HTTP protocol, each time a client retrieves a new Web page, the client actually opens a new and separate connection to the Web server. The server does not automatically maintain the contextual information about the client.

Advantages and Disadvantages of Stateless Protocol

The stateless protocol simplifies the server design, as there is no need for dynamic storage allocation for progressive conversation.

The disadvantage of statelessness is that it may be necessary to include additional information in every request, and this extra information will need to be interpreted by the server.

In-Class Question:

After you finish explaining Protocol and its features, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



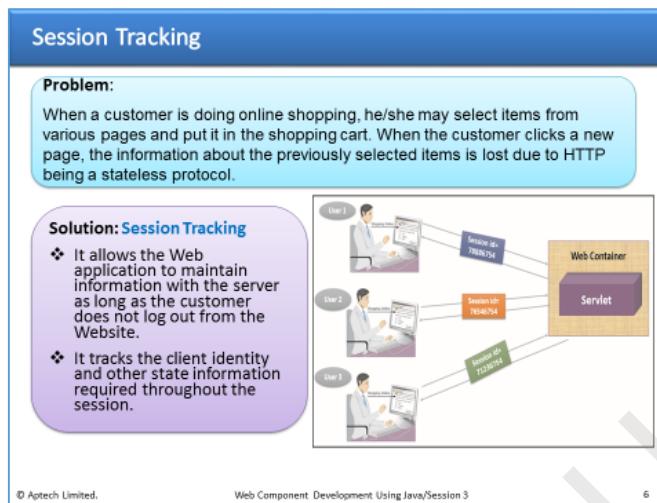
In which protocol the configuration setting, transaction, and information are not tracked and the protocol connections last for only one transaction?

Answer:

HTTP Protocol.

Slide 6

Let us understand session tracking mechanism.



Using slide 6, explain the session tracking mechanism.

Tell the students that Session Tracking helps to identify visitor sessions and the length of the visitor session. Session Tracking is a method to maintain data or state of a user. It is also known as Session Management. A single visitor session is a session from the time the user logs into the site and it lasts till the user logs out from the specific site. A session is a conversation between the host server and a client.

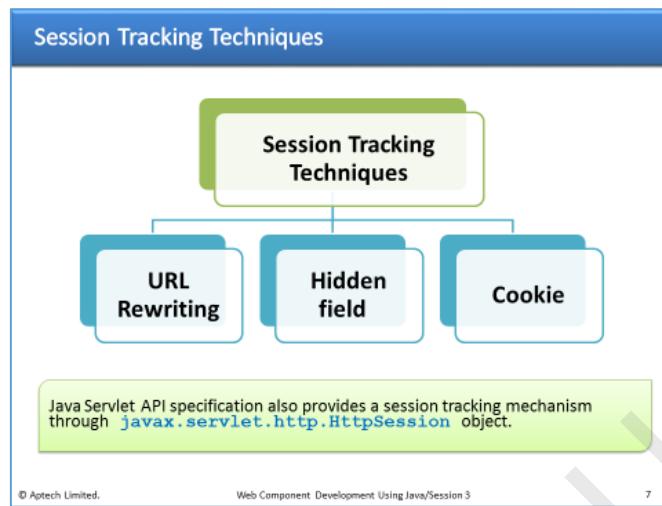
The example of session tracking can be a Shopping cart Web site where the user information is tracked in all the Web pages, till the user finishes the shopping of products and makes payment. The user can choose a product and add it to the shopping cart. When the user moves to a different page, the details in the shopping cart are still retained, so that the user can check the items in the shopping cart and then place the order.

Other examples where session tracking is implemented are as follows:

- Messenger service in which the username and password are saved and automatically displayed as soon as the user visits the Web site.
- Some Web sites track e-mail id of the users and automatically start sending the newsletters or updates on the mail.

Slide 7

Let us understand the different techniques used in session tracking.



Use slide 7 to explain the different techniques used in session tracking.

As HTTP protocol is a stateless protocol, so a Java Web application needs to maintain state of the client request at the Web server using session tracking techniques. The session tracking technique will help the applications to recognize the users in the multiple requests.

Tell the students that there are various techniques that can be used by a Web application to maintain a session which keeps track of data between the multiple requests from a client.

URL rewriting, Hidden field, and cookie are some of the techniques of Session Tracking. These are the most common techniques used by Web applications for maintaining clients' data.

Mention that Java Servlet API specification also provides a session tracking mechanism through `javax.servlet.http.HttpSession` object that is used by Servlet to store or retrieve information related to the user for maintaining session on the server.

Tips:

A Web container on the Java-enabled Web server associates a session with the user. Associating a session means creating a unique identifier for the user and passing it with the requests between the client and the server.

The Web container can pass the identifier and stores it on the client side for some duration through Cookies or pass it as a token in the URL with the request header.

Slide 8

Let us understand the URL rewriting technique.

URL Rewriting

- ❖ Uniform Resource Locator (URL) is the address of a resource located on the Web.
- ❖ The URL rewriting technique:
 - Adds some extra data at the end of the URL to identify the session.
 - Extra information can be in the form of extra path information or added parameters.
 - When the user clicks a link, the data from the page is appended after the '?' in the URL.
 - Is the lowest priority technique used for session management and is used as an alternative for cookies.

Original URL

Address: <http://www.apttech.com:8084/Servlet3Rewritten>

URL Rewritten with Extra Information

Address: <http://www.apttech.com:8084/Servlet3Rewritten?sessionId=10>

Original URL is appended with the parameter sessionId=10 at the end

This parameter is sent to the server as part of the client's request and helps the server to identify the client.

© Aptech Limited. Web Component Development Using Java/Session 3 8

Use slide 8 to explain the URL Rewriting technique.

Tell the students that if the setting to receive cookies at the client-side is disabled in the Web browser, the URL rewriting is used. URL Rewriting is a way to support anonymous Session Tracking.

It can be done by including the following:

1. Extra Path Information
 - Works on all servers
 - Works as target for GET and POST method
2. Added Parameters
 - Works on all servers
 - Fails as target for POST method
 - Parameter naming collision occurs
3. Custom Server-Specific URL change
 - Works on all servers that accept URL change

Slides 9 to 12

Let us understand how to pass information in URL.

Information in URL 1-4

- Figure shows a parameter or token attached at the end of the query string sent in the request.

The token consist of name-value pair.

© Aptech Limited. Web Component Development Using Java/Session 3 9

Information in URL 2-4

- The code snippet demonstrates the working of URL rewriting technique using two servlets.

```
/* Servlet1.java */
public class Servlet1 extends HttpServlet {
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException,
                         IOException {
        response.setContentType("text/html;charset=UTF-8");
        String sessionId = "Session1";
        PrintWriter pw = response.getWriter();
        pw.println("<html>");
        pw.println("<head></head>");
        pw.println("<body>");
        pw.println("Please click the below link:<br>");
        pw.println("<a href=/MyWebApp/Servlet2?sessionId=" + sessionId + ">View Report</a><br>");
        pw.println("</body>");
        pw.println("</html>");
    }
}
```

© Aptech Limited. Web Component Development Using Java/Session 3 10

Information in URL 3-4

```
/*Servlet2.java*/
public class Servlet2 extends HttpServlet {
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException,
                         IOException {
        response.setContentType("text/html;charset=UTF-8");
        String sessionId = request.getParameter("sessionId");
        PrintWriter pw = response.getWriter();
        pw.println("<html>");
        pw.println("<head></head>");
        pw.println("<body>");
        pw.println("This is the Session ID of the last page:");
        pw.println("<br>");
        pw.println("SessionID=" + sessionId + "<br>");
        pw.println("</body>");
        pw.println("</html>");
    }
}
```

© Aptech Limited. Web Component Development Using Java/Session 3 11

Information in URL 4-4

- ❖ In the code, **Servlet1** generates a hyperlink element which when clicked by the user will send the request to **Servlet2**.
- ❖ A name-value parameter **sessionId=Session1** is appended with the URL link specified with the `<a>` element.
- ❖ **Servlet2** accesses the parameter using `getParameter()` method and displays its value on the page.
- ❖ **Output:**

© Aptech Limited. Web Component Development Using Java/Session 3 12

Use slides 9 to 12 to explain how to pass information in URL.

Explain them that in URL rewriting, we append the URL of the Servlet with the parameters. The parameter contains a name/value pair which has the following format:

url?name1=value1&name2=value2&??

In the syntax, a name and a value is separated using an equal = sign. A parameter containing the name/value pair is separated from another parameter using the ampersand (&) symbol.

When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server for the processing.

Then, explain the code snippet mentioned on slides 10 and 11 to demonstrate how to use URL rewriting technique between the two servlets. Tell them that in the code, the **Servlet1** generates a hyperlink element which when clicked by the user will send the request to **Servlet2**. Then, the **Servlet2** access the parameter using `getParameter()` method and displays its value on the Web page as shown on slide 12.

Slide 13

Let us understand the advantages and disadvantages of URL rewriting.

URL Rewriting - Advantages and Disadvantages

Advantages

- Can be appended when sending the data from the HTML form.
- Can be sent along with the dynamic generated content from a Servlet.
- Is a preferred way to maintain the session when the browser doesn't support cookies or user disables the support for cookies.

Disadvantages

- URL can only be send through hyperlinks on the Web page.
- Long URLs cannot store that much information because of the URLs length limitation.
- URLs containing data information is visible, so it is not safe to be shared with others.

© Aptech Limited. Web Component Development Using Java/Session 3 13

Use slide 13 to explain the advantages and disadvantages of URL rewriting.

Usually, URL rewriting technique is used when information that is to be transferred is not very critical because the URL can be intercepted easily during transfer. For example, in an online shopping portal, a servlet can modify a URL to include user information, such as a username. The servlet can then display the URL.

Explain them that the extra appended URL information works with HTML forms sending request through GET method. However, the URL rewriting techniques fails with the POST method.

In-Class Question:

After you finish explaining URL rewriting technique, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which session tracking technique is the lowest priority technique used for session management and used as an alternative for cookies?

Answer:

URL Rewriting

Slides 14 to 16

Let us understand Hidden Form Fields technique.

Hidden Form Fields 1-3

- ❖ **Hidden fields:**
 - They are placed within an HTML form.
 - They are either a part of the static HTML form or dynamic form generated through Servlets.
 - They can be used to hold any kind of data.
 - They are not visible to the user and hence are not interpreted by the browser.
- ❖ **Advantages:**
 - User can pass much more information to the server.
 - Character encoding is not necessary.
- ❖ **Syntax:**

```
<INPUT TYPE="HIDDEN" NAME="..." VALUE="...>
```

© Aptech Limited. Web Component Development Using Java/Session 3 14

Hidden Form Fields 2-3

- ❖ The code snippet demonstrates the use of hidden field.

```
<!-- index.html -->
<body>
<form action = "MyServlet" method="POST">
<input type="hidden" name="job" value="Developer"/>
Name: <input type="text" name="firstname"/>
<input type="submit" name="Submit"/>
</form>
</body>
. . .

/* MyServlet.java */
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
```

© Aptech Limited. Web Component Development Using Java/Session 3 15

Hidden Form Fields 3-3

```
try {
    // Retrieves the name entered on the form
    String name = request.getParameter("firstname");
    // Retrieves the value from the hidden field
    String job = request.getParameter("job");
    out.println("<h3> Welcome: " + name + "<br>");
    out.println("\n Your Job: " + job + "</h3>");
    out.close();
} finally {
    out.close();
}
. . .
```

- ❖ **Output:**

© Aptech Limited. Web Component Development Using Java/Session 3 16

Use slides 14 to 16 to explain Hidden Form Field technique.

Tell the students that Hidden Form Fields technique is used when the form field is not to be displayed to the user. It is similar to URL rewriting, instead of appending values to the URL, hidden fields are placed within an HTML form.

You can use hidden form fields to maintain the session information of a user, while the user interacts with the Web application. A hidden form field is embedded in an HTML page and is not visible when viewed in a browser.

Mention that each time a servlet receives the form data, it would read all the parameters passed to it from the form along with the values in the hidden fields received from the HTML page.

Then, explain the code snippet given on slides 15 and 16 to explain the working of Hidden Fields in the session tracking.

Tips:

For a Servlet, there is no difference between the hidden fields and the other fields of the HTML form. To access the hidden field value on the Servlet, the `HttpServletRequest` object method named `getParameter()` is used.

Slide 17

Let us understand the advantages and disadvantages of Hidden Form Fields.

Hidden Form Fields - Advantages and Disadvantages

Advantages

- Supported in all browsers
- No special server requirements from clients
- Not visible directly to the user
- Works with or without cookies

Disadvantages

Works only when the page receives request through a submission of a form

© Aptech Limited. Web Component Development Using Java/Session 3 17

Use slide 17 to explain the advantages and disadvantages of Hidden Form Fields.

Tell them that the Hidden Field technique will not work if the user accesses a static Web page.

In-Class Question:

After you finish explaining Hidden form field and its features, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which session tracking technique is not visible to the user and hence not interpreted by the browser?

Answer:

Hidden Form Fields

Slide 18

Let us understand Cookies.

Cookies

- ❖ **Cookie:**
 - Small piece of information sent by a server to the client Web browser.
 - Stored on client's machine and is read back by the server on receiving request for the same page.
 - Contains one or more name-value pairs which are exchanged in request and response headers.
 - Stored for a limited life span on client's machine.
 - Are automatically deleted after a specified time period is completed.
 - Value of the cookie can uniquely identify a client.
- ❖ **HTTP request header contains:**
 - Information such as method, URL path, and HTTP protocol version.
- ❖ **HTTP response header contains:**
 - Date, size, and type of the file that server is sending back to the client.

© Aptech Limited. Web Component Development Using Java/Session 3 18

Use slide 18 to explain cookies.

Cookies are small text files that are stored by an application server in the client browser to keep track of all the users. A cookie has values in the form of name/value pairs. For example, a cookie can have a name, 'user' with the value, 'John'. They are created by the server and are sent to the client with the HTTP response headers.

Tell the students that when the server sends a cookie, the client receives the cookie, saves the cookie on the local hard disk, and sends it back to the server along with the HTTP request headers, each time the client accesses a page on that server.

Cookies are stored for a limited life span on the client's machine and once the specified time period is completed, they are automatically deleted. Explain that HTTP request header contains information such as method, URL Path, and HTTP protocol version. Similarly, HTTP response header contains data, size, and type of the file that server sends back to the client.

Cookies are of two types:

1. Session cookies
 - Stored in memory
 - Lost when user exits the Web application
2. Permanent cookies
 - Stored in persistent storage
 - Lost when they expire

Slide 19 to 27

Let us understand the Cookie API.

Cookie API 1-9

- Figure depicts the concept of cookie.

As the value of the cookie can uniquely identify a client, cookies are commonly used in session tracking.

- Drawback:**
 - Most browsers allow the users to deactivate cookies.

© Aptech Limited. Web Component Development Using Java/Session 3. 19

Cookie API 2-9

- The code snippet demonstrates how to create and add cookie in the Servlet response.

```
//This snippet remember an added item by adding to a cookie
public void doGet (HttpServletRequest request,
HttpServletResponse response) throwsServletException,
IOException
{
    //If the user wants to add an item in a cookie
    if (values != null) {
        ItemId = values[0];
        Cookie getItem = new Cookie("Buy", ItemId);
        getItem.setComment("User has indicated a desire "
+ "to buy this book from the bookstore.");
        response.addCookie(getItem);
    }
}
```

© Aptech Limited. Web Component Development Using Java/Session 3. 20

Cookie API 3-9

- The Servlet API provides `javax.servlet.http.Cookie` class for creating and working with cookies.
- The `Cookie` class provides several methods which help in cookie management:
 - public void setMaxAge(int expiry)**
 - This method sets the maximum age of the cookie in seconds.
 - If the value is positive, then the cookie will expire after that many seconds which is specified by the expiry.
 - For example, `demoCookie.setMaxAge(600);`
 - public int getMaxAge()**
 - It returns the maximum age of the cookie.
 - It returns an integer which specifies the maximum age of the cookies in seconds.

© Aptech Limited. Web Component Development Using Java/Session 3. 21

Cookie API 4-9

- The code snippet demonstrates how to get the cookie age.

```
/* Prints the cookie age */
PrintWriter out = response.getWriter();
Cookie demoCookie = new Cookie("FavColor", "Blue");
demoCookie.setMaxAge(600);
int result = demoCookie.getMaxAge();
out.println("Cookie Age: " +result);
. . .
```

- The code returns the maximum age of the cookie, which is specified in seconds.

- public void setValue(java.lang.String newValue)**
 - This method assigns a new value to a cookie after the cookie is created.

Cookie API 5-9

- The code snippet demonstrates how to set the value of the cookie.

```
// Sets the value of the cookie
public void setCookieValue(String value)
{
    if (value == null || (value.equals("")))
        throw new IllegalArgumentException("Invalid
cookie value set in: " + getClass().getName());
    if (cookie != null)
        cookie.setValue(value);
}
```

- public java.lang.String getValue()**
 - Returns a string containing the cookie's present value.

Cookie API 6-9

- public java.lang.String getName()**
 - Returns the name of the cookie.
 - Once, the cookie has been created its name cannot be changed.
 - The code snippet demonstrates how to retrieve the name and value of the cookie.

```
// Retrieves the name of the cookie and its value
for (int i = 0; i<cookies.length; i++) {
    String name = cookies[i].getName();
    String value = cookies[i].getValue();
    out.println("name = " + name +"; value = " +
value);
```

- public void setPath(String uri)**
 - Sets the path for the cookie.
 - Is available to all the pages specified in the directory and its subdirectories.

Cookie API 7-9

- The code snippet demonstrates how to set the path for the cookie.

```
// Sets the path for the cookie
cookie = new Cookie("sessionId", "erased");
cookie.setPath("/servlet/SessionCookie");
resp.setHeader("Set-Cookie", cookie.toString());
```

public java.lang.String getPath()

- Returns the path on the server to which the client returns the cookie.
- Returns a string specifying a path that contains a servlet name.
- Is available to all sub paths on the server.

For example, /AptechDemo

© Aptech Limited.

Web Component Development Using Java/Session 3

25

Cookie API 8-9

public Cookie[] getCookies()

- Reads the cookies from a request by using the `HttpServletRequest.getCookies()` method.
- Returns an array containing all of the `Cookie` objects that the client sends with the request.
- The code snippet demonstrates how to read cookies received in the client request.

```
// Retrieves cookies from the request object
Cookie[] cookies = request.getCookies();
// Iterates through the array
for(int i=0; i<cookies.length; i++) {
    Cookie = cookies[i];
    // Print cookie details
    out.println("Cookie Name: " + cookie.getName());
    out.println("Cookie Value: " +
    cookie.getValue());
}
```

© Aptech Limited.

Web Component Development Using Java/Session 3

26

Cookie API 9-9

void addCookie (Cookie cookie)

- Is Sent using `HttpServletResponse` object.
- Adds field to the HTTP response headers to send cookies to the browser, one at a time.
- Adds specified cookie to the response and can be called multiple times to set more than one cookies.

For example, `response.addCookie(new Cookie("cookiename", "cookievalue"));`

© Aptech Limited.

Web Component Development Using Java/Session 3

27

Use slides 19 to 27 to explain the Cookie API.

The `Cookie` class of `javax.servlet.http` package represents a cookie. The `Cookie` class provides a constructor that accepts the name and value to create a cookie.

The following code snippet shows the constructor for creating a cookie with name and value pair:

```
public Cookie(String name, String value);
```

The `HttpServletResponse` interface provides the `addCookie()` method to add a cookie to the response object, for sending it to the client.

The following code snippet shows how to add a cookie:

```
resp.addCookie(cookie);
```

The `HttpServletRequest` class provides the `getCookies()` method that returns an array of cookies that the request object contains.

The following code snippet shows how to retrieve cookies:

```
Cookie cookie[] = req.getCookies();
```

The `Cookie` class provides several methods to set and retrieve various properties of a cookie.

The following table lists the several methods that help in cookie management:

Method	Description
<code>public void setMaxAge(int expiry)</code>	Sets the maximum age of the cookie in seconds for which the client browser can retain the cookie value.
<code>public int getMaxAge()</code>	Returns the maximum age of the cookie in seconds.
<code>public java.lang.String getValue()</code>	Returns the value of the cookie. The method returns a string containing the cookie's present value.
<code>public java.lang.String getName()</code>	Returns the name of the cookie. Once, the cookie has been created its name cannot be changed.
<code>public void setPath(String uri)</code>	Sets the path for the cookie. The cookie is available to all the pages specified in the directory and its subdirectories.
<code>public java.lang.String getPath()</code>	Returns the path on the server to which the client returns the cookie.

Explain each method of the `Cookie` class with a code snippet mentioned on slides 22 to 27.

Slides 28 to 31

Let us understand securing cookies.

Securing Cookies 1-4

- ❖ **Problems with Cookies:**
 - JavaScript can be used to access the cookies from the machine.
 - Cookies are available across scripts and may be accessed by hackers for manipulation.
- ❖ To secure the cookies from hackers on the Web, user can configure cookies with two security settings namely, `secure` and `HttpOnly`.
- ❖ **`secure` flag:**
 - Informs the Web browser that cookies should be sent only on the SSL connection.
- ❖ **`HttpOnly` flag:**
 - Informs the browser that the content of the cookie are not accessible within JavaScript.
 - Is included in the HTTP response header and prevents the cookies from certain kind of attacks.

© Aptech Limited. Web Component Development Using Java/Session 3 28

Securing Cookies 2-4

- ❖ The `Cookie` class provides the following methods:
 - `public void setSecure()` • This method informs the browser to send the cookie only through secured protocol, such as HTTPS.
 - `public void setHttpOnly(boolean)` • This method can be used to mark or unmark the cookie.
• If set as true, then cookie is marked as `HttpOnly` and are not exposed to client-side scripting code.
 - `public boolean isHttpOnly()` • This method is used to check whether the cookie has been marked as `HttpOnly`.

© Aptech Limited. Web Component Development Using Java/Session 3 29

Securing Cookies 3-4

- ❖ The code snippet shows how to set the cookies as `HttpOnly` and `secure` flag programmatically in servlet.

```
protected void doGet(HttpServletRequest req,
HttpServletResponse res) throws ServletException,
IOException {
    ...

    Cookie = new Cookie("Color", "Cyan");
    Response.addCookie(cookie);
    cookie.setHttpOnly(true);
    cookie.setSecure(true);

    ...
    boolean status = cookie.isHttpOnly();
    out.println("<br>Status of Cookie - Marked as HttpOnly
= " + status);
}
```

© Aptech Limited. Web Component Development Using Java/Session 3 30

Securing Cookies 4-4

- ❖ Alternatively, the user can provide the declaration settings in the web.xml configuration file.
- ❖ The code snippet shows the web.xml file with security setting for a cookie.

```
. . .
<session-config>
  <cookie-config>
    <http-only>true</http-only>
    <secure>true</secure>
  </cookie-config>
</session-config>
. . .
```

© Aptech Limited. Web Component Development Using Java/Session 3 31

Use slides 28 to 31 to explain securing cookies.

Tell the students that secure flag informs the Web browser that cookies should be sent only on the Secure Socket Layer (SSL) connection. This means any page of the Web application that is not secured will not be able to access cookies.

Mention that Servlet 3.0 provides an `HttpOnly` flag which informs the browser that the content of the cookie are not accessible within JavaScript. It is included in the HTTP response header and prevent the cookies from certain kinds of attacks. Alternatively, the declaration settings for security can be done in the `web.xml` file.

The cookie class provides the following methods:

- `public void setSecure ()` - This method informs the browser to send the cookie only through secured protocol, such as HTTPS.
- `public void setHttpOnly(boolean)` - This method can be used to mark or unmark the cookie. If set as true, then cookie is marked as `HttpOnly` and are not exposed to client-side scripting code.
- `public boolean isHttpOnly()` - This method is used to check whether the cookie has been marked as `HttpOnly`.

Then, explain the code snippets given on slides 30 and 31, the cookies as `HttpOnly` and secure flag programmatically in servlet and providing the settings in the `web.xml`.

In-Class Question:

After you finish explaining cookies and its features, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



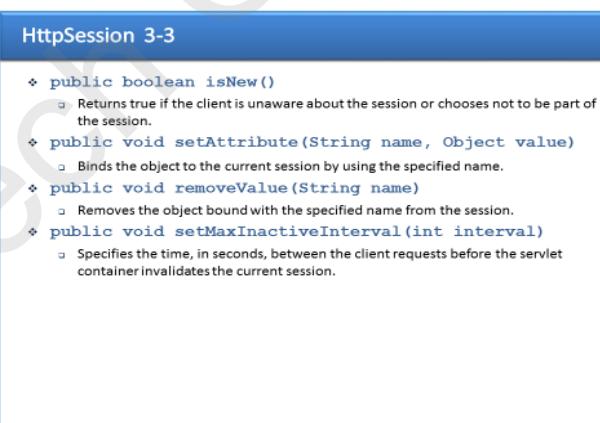
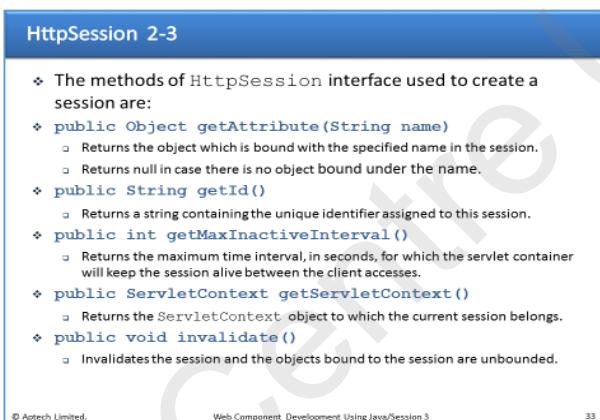
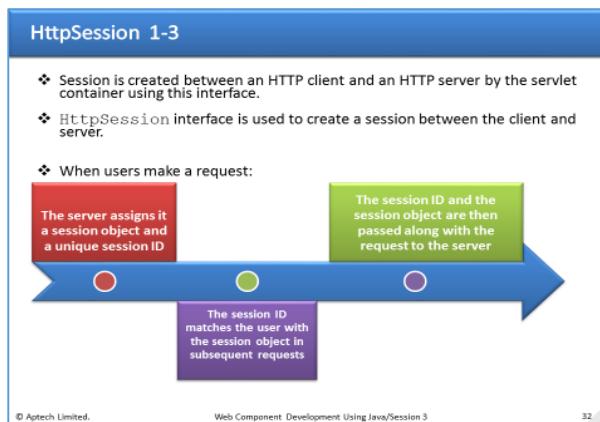
Which method in cookie class informs browser to send the cookie only through secured protocol such as HTTPS?

Answer:

`public void SetSecure()`

Slides 32 to 34

Let us understand the HTTP Session.



Use slides 32 to 34 to explain HTTP session.

HTTP session provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user. HTTP session can be used to bind objects and to view and manipulate information about a session.

Tell the students that there are various methods of HTTP Session interface used to create a session as listed on slides 33 and 34.

Slides 35 and 36

Let us understand storing information in a Session.

Storing Information in a Session 1-2

- ❖ The data can be stored in an `HttpSession` object using the name-value pairs.
- ❖ The data which is stored is available throughout the current session.
- ❖ The method `setAttribute()` is used to store the data in a session.
- ❖ The code snippet demonstrates how to create a new session object and set object in it.

```
public void doGet( HttpServletRequest request,
HttpServletResponse ) throws IOException,
ServletException
{
    HttpSession httpSession = request.getSession(true);
    // Gets current session or create a new one if not exist
    if (httpSession.isNew())
    {
        // Set the maximum interval for the session
        httpSession.setMaxInactiveInterval(60);
    }
}
```

© Aptech Limited. Web Component Development Using Java/Session 3 35

Storing Information in a Session 2-2

```
// Sets the name attribute name as Jami
httpSession.setAttribute("name", "Jenny");
// Sets the attribute background colour to "#FFFFFF"
httpSession.setAttribute("age", new Integer(20));

}
```

- ❖ In the code,
 - `getSession()` returns the current session object associated with the request. If the session does not exists, then the boolean value `true` indicates to create a new session.
 - `isNew()` returns a boolean value indicating whether it is a new session or not. If it returns `true`, then the objects are bounded to the new session through `setAttributes()` method.
 - `setMaxInactiveInternal()` specifies the time between the requests from the client before the servlet container invalidates the session.

© Aptech Limited. Web Component Development Using Java/Session 3 36

Use slides 35 and 36 to explain that data can be stored in `HttpSession` object as name value pair using `setAttribute` and this data is available throughout the current session.

Then, explain the code snippet given on slide 36 to create a new session object and set object in it. Tell them that in the code, `getSession()` returns the current session object associated with the request. If the session does not exists, then the boolean value `true` indicates to create a new session.

The `isNew()` returns a boolean value indicating whether it is a new session or not. If it returns `true`, then the objects are bounded to the new session through `setAttributes()` method.

The method `setMaxInactiveInternal()` specifies the time between the requests from the client before servlet container invalidates the session.

Slide 37

Let us understand retrieving information stored in a session.

Retrieving Information Stored in a Session

- ❖ The code snippet explains the procedure for retrieving name and age stored in the session.

```
// Retrieves name and age from session
String myText = (String)session.getAttribute("name");
int myNumber = ((Integer)
session.getAttribute("age")).intValue();
```
- ❖ The stored values in the session can be retrieved using `getAttribute()` method.
- ❖ Since, the return type is an object, typecasting of data associated with that attribute name in the session is done.

© Aptech Limited. Web Component Development Using Java/Session 3 37

Use slide 37 to explain retrieving information stored in a session.

Tell the students that using `getAttribute()` method, the user can retrieve the values stored in the session. Since, the return type is an object, typecasting of data associated with that attribute name in the session is done. If the return value is null, it can be concluded that no such attribute exists. Therefore, it is required to check for null, before calling the methods on the object.

Slide 38

Let us understand invalidating a Session.

Invalidate a Session

- ❖ The invalidate() method is used to avoid the hacker from causing any harm to the Web application.
- ❖ It destroys the data in a session that another servlet or JSP might require in future.
- ❖ Sessions should be invalidated cautiously as they are associated with the client, not with individual servlets or JSP pages.
- ❖ The code snippet demonstrates invalidating the session.

```
// Returns current session or a new session if it does not exist
HttpSession session = request.getSession (true);
// Checks the session
if (session.isNew() == false) {
// Invalidates the session if it is not a new session
session.invalidate();
// Creates a new session
session = request.getSession (true);
}
```

- ❖ The code directs the session to invalidate itself if it is not created newly.
- ❖ To invalidate the session manually, the invalidate() method should be called.

© Aptech Limited. Web Component Development Using Java/Session 3 38

Use slide 38 to explain invalidating a session.

Tell the students to avoid the unauthorized hacking, the invalidate() method is called. It destroys the data in a session that another servlet or JSP might require in future. Session can be invalidated through code or manually using invalidate() method. Sessions are invalidated for security and memory management.

Then, explain the code snippet as shown on slide 38 to explain how to invalidate the session.

Slide 39

Let us understand session timeout.

Session Timeout

- ❖ After a certain time period of inactivity the session is destroyed to prevent the number of sessions increasing infinitely.
- ❖ It happens if the user remains inactive for a period greater than the set inactive time period.
- ❖ The session timeout period can be set either in the web.xml file or can be set by the method `setMaxInactiveInterval()`.
- ❖ The setting for session time-out should be written in web.xml file.
- ❖ **Syntax:**

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

- N in the fragment is session timeout period.

© Aptech Limited. Web Component Development Using Java/Session 3 39

Use slide 39 to explain session timeout.

Tell the students that session timeout is necessary as a session utilizes the memory locations to store information and long period of inactivity will occupy the memory unnecessarily. After a certain time period of inactivity, the session is destroyed to prevent the number of sessions increasing infinitely.

The session timeout period can be set either in the `web.xml` file or can be set by the method `setMaxInactiveInterval()`. This method is used to specify the time between the requests from the client before servlet container invalidates the session.

Then, explain the code snippet given on slide 39 to set the session timeout in `web.xml` file.

In-Class Question:

After you finish explaining session timeout, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which method can be used to set the timeout for the session?

Answer:

`setMaxInactiveInterval()`

Slide 40

Let us summarize the session.

Summary

- ❖ Session tracking allows the server to keep track of successive requests made by the same client.
- ❖ Some of the session tracking techniques are namely, URL rewriting, hidden field, and cookie.
- ❖ Java Servlet specification provides a session tracking mechanism through javax.servlet.http.HttpSession object.
- ❖ The URL rewriting technique adds some extra data at the end of the URL to identify the session.
- ❖ Hidden form fields are used to pass data to the server-side resource invisibly from the user.
- ❖ A cookie is a small piece of information sent by a server to the client Web browser. The cookies are stored on client's machine and are read back by the server on receiving request for the same page.
- ❖ To secure the cookies from hackers on the Web, you can configure cookies with two security settings namely, secure and HttpOnly.
- ❖ The HttpSession interface is used to create a session between the client and server. The session is created between an HTTP client and an HTTP server by the servlet container using this interface.

© Aptech Limited. Web Component Development Using Java/Session 3 40

In slide 40, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

3.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should explore how to apply filters and annotations to the servlets.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 4 – Filters and Annotations

4.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

4.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the concept and working of filters
- List the benefits of filters
- Explain the Filter API interfaces and there methods
- Explain the use of wrapper classes in managing Servlets
- Explain how to alter a request and response using filters
- Describe the basic concept of annotations
- Explain the different types of annotations supported in Servlet API
- Explain the steps to upload the file using HTML form elements
- Explain the mechanism to upload files on the server using Servlet

4.1.2 Teaching Skills

To teach this session successfully, you should be aware of working of filters and how to create filters that can manipulate client requests and responses. You should also gain knowledge on annotations and its types supported in Java Servlets.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❖ Explain the concept and working of filters
- ❖ List the benefits of filters
- ❖ Explain the Filter API interfaces and there methods
- ❖ Explain the use of wrapper classes in managing Servlets
- ❖ Explain how to alter a request and response using filters
- ❖ Describe the basic concept of annotations
- ❖ Explain the different types of annotations supported in Servlet API
- ❖ Explain the steps to upload the file using HTML form elements
- ❖ Explain the mechanism to upload files on the server using Servlet

© Aptech Ltd. Web Component Development Using Java/Session 4 2

Tell them that they will be introduced to the concept and working of filter API. The session teaches how to create filters that can manipulate client requests and server responses. Further, the session provides a comprehensive study of annotations supported by Java Servlets and uploading of files using HTML form elements and servlets.

4.2 In-Class Explanations

Slide 3

Let us understand filters.

Introduction

- ❖ Java supports filter objects which performs processing of the request before it acquires the resource.
 - For example, data sent in the response from a Servlet must be encrypted to avoid any malfunction threat.
 - This needs an extra processing of encryption to be done on the received response from the Servlet.
- ❖ Before sending the response, filters perform manipulation of the responses sent to the client.
- ❖ Figure shows the processing of request and response done by filters.

© Aptech Ltd. Web Component Development Using Java/Session 4 3

Use slide 3 to introduce filter objects. Tell the students that there are some situations when data sent or received from the Web resources need to be processed with extra functionality. For example, a common scenario for processing payments sent in response over the internet. The data sent in the response from a servlet must be encrypted to avoid any malfunction threat. This needs an extra processing of encryption to be done on the received response from the servlet.

Java supports filter objects that vigorously intercepts requests from clients and responses from the server to modify or use the information contained in the requests or responses. Generally, filters do not generate responses by themselves, but instead offer common functions that can be devoted to any kind of JSP page or servlet. Filters also offer the capability to encapsulate repetitive tasks in reusable entities.

Some of the common scenarios of the Web applications where filters can be configured are as follows:

- Authenticating and blocking requests depending on user identification
- Conversion of images
- Scaling of maps
- Compression of data to manage downloads
- Tracking and auditing of users of a Web application
- Encrypting responses from the server
- Transformation of XML content
- Triggering resource access events
- Tracking and localized targeting of request and response

Use the figure given on slide 3 to display and explain the processing of request and the response by filters.

Slides 4 and 5

Let us understand the concept of filters.

Concept of Filter 1-2

- ❖ **Filters:**
 - Are Java classes.
 - Used to encapsulate the preprocessing and post processing functionalities common to a group of Servlets or JSP pages.
 - Were introduced as a Web component in Java Servlet specification.
 - Reside in the Web container along with the other Web components, such as Servlet or JSP pages.
- ❖ Figure depicts filters in a Web application.

© Aptech Ltd. Web Component Development Using Java/Session 4 4

Concept of Filter 2-2

- ❖ Some of the important features provided by the filters to the Web applications are as follows:
 - Optimization of the time taken to send a response back to a client.
 - Compression of the content size sent from a Web server to a user over the Internet.
 - Conversion of images, texts, and videos to the required format.
 - Optimization of the available bandwidth on the network.
 - Authentication of the users in the secured Web site.
 - Encryption of the request and response header for addressing security concerns.

© Aptech Ltd. Web Component Development Using Java/Session 4 5

Use slides 4 and 5 to explain the concept of filters. Tell the students that a filter is a Java class that is invoked in response to a user request for a resource in a Web application. The Web application resources comprise static resources such as HTML pages or images, Java Server Pages, and Java Servlets.

Use the figure given on slide 4 to display the filter acting as interceptors for processing the request and response between the client and Web resources or end points, such as Servlet and JSP.

Use slide 5 to explain some of the important features provided by the filters to Web applications.

Slide 6

Let us understand filter chain.

Filter Chaining

- ❖ There can be more than one filter between the client and the Web resources, thus forming a filter chain.
- ❖ Figure shows chaining of the filters.

A request or a response is passed through one filter to the next in the filter chain.
Each request and response is serviced by filters configured in the chain, before the Servlet is called by the Web container and after the Servlet responds.

© Aptech Ltd. Web Component Development Using Java/Session 4 6

Use slide 6 to explain filter chain. Use the figure given on slide 6 to display and explain that how the filters are chained.

Tell them there can be more than one filter between the client and the Web resources, thus forming a filter chain. A request or a response is passed through one filter to the next in the filter chain. So each request and response has to be serviced by filters configured in the chain, before the Servlet is called by the Web container and after the Servlet responds.

Slides 7 and 8

Let us understand the working of filters.

Working of Filters 1-2

- ❖ A typical filter operates in the following way:

The Web container instantiates the filters and loads them for preprocessing.

The filter intercepts the request from the user to the servlet.

After the request is processed, the filter can do the following:

- Generate response and return to the client.
- Pass the modified or unmodified request to other filter in the chain.
- If the request is received by the last filter in the chain, it passes the request to the destination Servlet.
- It may route the request to a different resource rather than the associated Servlet.

Then, the filter sends the serviced request to the appropriate Servlet.

© Aptech Ltd. Web Component Development Using Java/Session 4 7

Working of Filters 2-2

- ❖ When the response is returned to the client, the response passes through same set of filters in reverse order.
- ❖ Figure depicts working of a typical filter.

© Aptech Ltd. Web Component Development Using Java/Session 4 8

Use slides 7 and 8 to explain the typical working of filters. Tell the students that filters are defined in the context of a Web application. A filter intercepts a user request for a particular resource or a group of resources based on a URL pattern and implements the code in the filter. A single filter or multiple filters can be specified to be invoked in a particular order, called a chain, for each resource or group of resources.

Filters dynamically access incoming requests from the user before the servlet processes the request. Filters can also access the outgoing response from the Web resources before it reaches the user. The information about the configured filters is provided when the application is deployed on the server through `web.xml`. When the request for the Web resource is sent to the Web server, the Web container instantiates the filters and loads them for preprocessing.

Use the figure given on slide 8 to explain the working of a typical filter.

Tell them that a typically filter operates as mentioned:

1. The filter intercepts the request from the user to the servlet.
 - a. After the request is processed, the filter can generate response and return to the client.
 - b. Pass the modified or unmodified request to other filter in the chain.
 - c. If the request is received by the last filter in the chain, it passes the request to the destination Servlet.
 - d. It may route the request to a different resource rather than the associated Servlet.
2. Then, the filter sends the serviced request to the appropriate Servlet.
3. Similarly, when the response is returned to the client, the response passes through same set of filters in reverse order.

Slides 9 and 10

Let us understand usage of filters.

Usage of Filters 1-2

- ❖ Some of the filter categories are as follows:

Authentication filters	Allows the users to access any resource in secured Websites after providing proper username and password.
Logging and auditing filters	Tracks the activities of users on a Web application and log them.
Image conversion filters	Scales the image size or change the image type as per requirements.
Data compression filters	Helps in compressing uploaded or downloaded file size, thereby reducing the bandwidth requirement and time for downloading.
Encryption filters	Helps in encrypting the request and response header.

© Aptech Ltd. Web Component Development Using Java/Session 4 9

Usage of Filters 2-2

Filters that trigger resource access events	Helps in registering the particular database for a Web application before they retrieve or manipulate data according to the request made by the client.
XML transformation filters	Reads the XML data from the response and applies an XSLT transformation before sending it to the client browser.

- ❖ Figure depicts a compression filter.

```

graph LR
    A[Data Packet  
Size 50 MB] --> B[Compression Filter]
    B --> C[Data Packet  
Size 10 MB]
    C --> D[WEB-RESOURCE]
  
```

© Aptech Ltd. Web Component Development Using Java/Session 4 10

Use slides 9 and 10 to explain the usage of few of the filter categories.

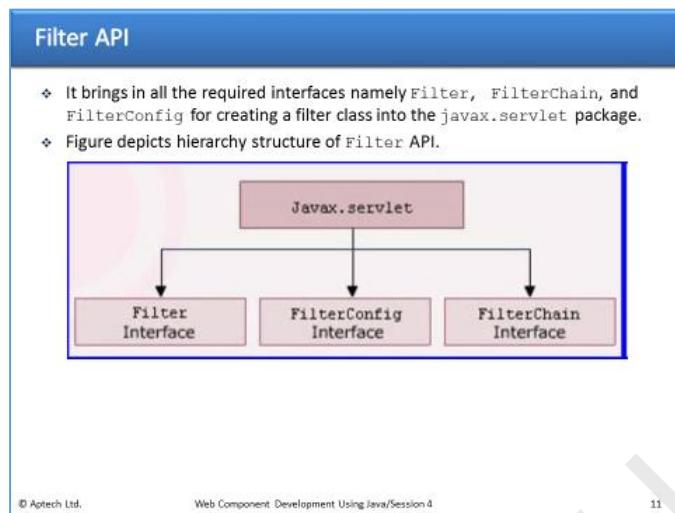
Tell the students that filters are categorized according to the services they provide to the Web applications. Some of the filters are authentication filters, data compression filters, encryption filters, image conversion filters, MIMI-TYPE chain filters, logging and auditing filters, and so on.

Filters are configured in the deployment descriptor `web.xml` file. They are then mapped to either servlet names or URL patterns in the deployment descriptor of the Web application.

Use the figure given on slide 10 to display and describe the working of data compression filter.

Slide 11

Let us understand Filter API.



Use slide 11 to explain filter API. Tell the students that filter API is part of Servlet API and the filter interface is found in the `javax.servlet` package.

Use the figure as shown on slide 11 to depict hierarchy structure of Filter API. Tell the students that a typical filter needs to implement the `Filter` interface and the methods contained in it.

Slides 12 to 15

Let us understand filter interface.

Filter Interface 1-4

- ❖ The Filter interface is part of the Servlet API.
- ❖ The Filter interface provides the life cycle methods of a filter. It must be implemented to create a filter class.
- ❖ The methods of the Filter interface are as follows:
 - **init()**
 - It is called by the servlet container to initialize the filter.
 - It is called only once.
 - The init() method must complete successfully before the filter is asked to do any filtering work.
 - **Syntax:**

```
public void init(FilterConfig filterConfig)
throws ServletException
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

12

Filter Interface 2-4

- **doFilter()**
 - It is called by the container each time a request or response is processed.
 - It then examines the request or response headers and customizes them as per the requirements.
 - It also passes the request or response through the FilterChain object to the next entity in the chain.
- **Syntax:**

```
public void doFilter(ServletRequest req,
ServletResponse res, FilterChain chain) throws
IOException, ServletException
```
- **destroy()**
 - It is called by the servlet container to inform the filter that its service is no more required.
 - It is called only once.
- **Syntax:**

```
public void destroy()
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

13

Filter Interface 3-4

- Figure shows the template of a class implementing the Filter interface.

```
public class MyGenericFilter implements javax.servlet.Filter {
    public FilterConfig filterConfig;
    public void init(FilterConfig filterConfig) {
        this.filterConfig = filterConfig;
    }
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        chain.doFilter(request, response);
    }
    public void destroy() {
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

14

Filter Interface 4-4

- The code snippet demonstrates the implementation of filter to display a message before the invocation of the configured Servlet.

```
public class MessageFilter extends Filter {
    private FilterConfig config;
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        System.out.println("Invoking Filter");
        System.out.println("Preprocessing done by filter");
        chain.doFilter(request, response);
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

15

Use slides 12 to 15 to explain Filter interface and its methods.

Using slide 12, explain the syntax of `init()` method. Tell them that in the syntax, `filterConfig` is the object of `FilterConfig` interface passed as a parameter to the `init()` method and `ServletException` is a general exception that can be thrown by a Servlet.

Using slide 13, explain the syntax of `doFilter()` and `destroy()` methods.

The syntax of `doFilter()` method:

```
public void doFilter(ServletRequest req, ServletResponse res,
    FilterChain chain) throws IOException, ServletException
```

In the syntax, `ServletRequest` defines an object to provide client request to a servlet, `ServletResponse` defines an object to help the Servlet in sending response to the user, and `FilterChain` interface creates an object chain to invoke next filter component in the chain.

Use the figure given on slide 13 to show the template of a class implementing the `Filter` interface. Use the code snippet given on slide 14 to demonstrate the implementation of filter to display a message before the invocation of the configured Servlet. Tell them that in the code, the `MessageFilter` class implements the `Filter` interface. When the filter is invoked, the container will pass the `request`, `response`, and `chain` objects to the `doFilter()` method. These objects are passed to the next component in the chain, and it can be a filter or a Servlet as shown on slide 15.

Slide 16

Let us understand configuring filter.

Configuring Filter

- ❖ The configuration of a filter inside the `web.xml` file ensures that filter is the part of that application.
- ❖ In the deployment descriptor, filter is declared by the `<filter>` element.
- ❖ It defines a filter class and its initialization parameters.
- ❖ Following elements can be defined within a filter element:
 - `<display-name>`
 - `<description>`
 - `<filter-name>`
 - `<filter-class>`
 - `<init-param>`

© Aptech Ltd. Web Component Development Using Java/Session 4 16

Use slide 16 to explain configuring filters.

Tell the students that each filter is a Web component, which along with others forms a complete Web application. In addition to programming the filters, they have to be configured in the `web.xml` file as to how they are mapped to servlets when the application is deployed in a Web container.

List and explain the elements that can be defined within a filter element as:

- **`<display-name>`**
This element accepts the short name of the filter, which is intended to be displayed by GUI tools. It is not mandatory inside the `<filter>` element.
- **`<description>`**
This gives a short description about the filter.
- **`<filter-name>`**
This defines the name of the filter. This is used to refer to the filter definition somewhere in the deployment descriptor. This element is mandatory.
- **`<filter-class>`**
It defines the class name of the filter. This is a mandatory element in a filter definition.
- **`<init-param>`**
It holds the name and value pair of an initialization parameter. It is optional within the `<filter>` element.

In-Class Question:

After you finish explaining filter configuration, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which are the mandatory elements that can be defined within a filter element?

Answer:

`<filter-name>` and `<filter-class>`

Slides 17 and 18

Let us understand filter-mapping element.

Filter-mapping Element 1-2

- ❖ It specifies which filter to execute depending on a URL pattern or a Servlet name.
- ❖ This actually sets up a logical relation between the filter and the Web application for sequential control flow of request and response between them.
- ❖ Following elements are defined inside a filter-mapping element:
 -  `<filter-name>`
 - The element inside the deployment descriptor holds the name of the filter to which a URL pattern or a Servlet is mapped.
 - It is a mandatory element for mapping a filter.
 -  `<url-pattern>`
 - The element describes a pattern used to resolve URLs.
 - It is a mandatory element for mapping a filter.
 -  `<servlet-name>`
 - The element specifies the name of a servlet whose request and response will be modified by the filter.
 - It is a mandatory element.

© Aptech Ltd. Web Component Development Using Java/Session 4 17

Filter-mapping Element 2-2

- ❖ The code snippet shows the mapping of MessageFilter with the MyServlet in the web.xml file.

```
<filter>
  <display-name>Filter</display-name>
  <description>This is my first filter
  </description>
  <filter-name>Message</filter-name>
  <filter-class>
    servlet.filter.MessageFilter
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>Message</filter-name>
  <servlet-name>MyServlet</servlet-name>
</filter-mapping>
...
```

© Aptech Ltd. Web Component Development Using Java/Session 4 18

Use slide 17 to explain filter-mapping elements.

Use the code snippet given on slide 18 to display the mapping of MessageFilter with MyServlet in the web.xml file. Tell them that in the code, the mapped filter named Message will be executed when the mapped servlet named MyServlet is requested by the client. The `<servlet-name>` can be replaced with `<urlpattern>` element to match a specific URL pattern.

Slides 19 to 21

Let us understand passing of initialization parameters to filter.

Passing Initialization Parameters to Filter 1-3

- ❖ The `init()` method of a filter initializes the filter.
- ❖ It receives the object of `FilterConfig` object created by the Web container.
- ❖ Then, the initialization parameters are read in the `init()` using the methods of `FilterConfig`.
- ❖ Table lists some of the methods of `FilterConfig`.

Method	Description
<code>String getFilterName()</code>	Returns the name of the filter defined in the web.xml or deployment descriptor file in the Web application.
<code>String getInitParameter(String name)</code>	Returns the value of the named initialization parameter as a string. Returns null if the servlet has no attribute.
<code>Enumeration getInitParameterNames()</code>	Returns the names of the initialization parameter as an enumeration of String objects.
<code>ServletContext getServletContext()</code>	Returns the <code>ServletContext</code> object used by the caller to interact with its Servlet container and filter.

Passing Initialization Parameters to Filter 2-3

- ❖ The code snippet exhibits the use of the methods in the `FilterConfig` interface.

```
public class MessageFilter implements Filter {
    private String message;
    private FilterConfig config;
    public void init(FilterConfig config) throws
        ServletException {
        this.config = config;
        message = config.getInitParameter("message");
    }
    public void doFilter(HttpServletRequest request, final
        HttpServletResponse response, FilterChain chain)
        throws
        java.io.IOException, javax.servlet.ServletException {
        System.out.println("Entering MessageFilter");
        request.setAttribute("messageObj", message);
        chain.doFilter(request, response);
        System.out.println("Exiting MessageFilter");
    }
}
```

Passing Initialization Parameters to Filter 3-3

- ❖ Figure shows the configuration of filter with initialization parameters in the `web.xml` file.

```
<filter>
<filter-name>Message</filter-name>
<filter-class>servlet.filter.MessageFilter</filter-class>
<!-- Initialization parameters -->
<init-param>
    <param-name>message</param-name>
    <param-value>Welcome to Filter</param-value>
</init-param>
</filter>

<filter-mapping>
<filter-name>Message</filter-name>
<servlet-name>MyServlet</servlet-name>
</filter-mapping>
```

Use slides 19 to 21 to explain passing of initialization parameters to filter. Tell the students that some servlets and filters need initialization parameters. The initialization parameters for a particular filter is contained within the `init-param` element of that filter. These parameters are different

from the context parameters, and each of the `init-param` contains `param-value` element and a `param-name` element.

Use the table as shown on slide 19 to list and explain some of the methods of `FilterConfig`.

While explaining `Enumeration getInitParameterNames ()` method, tell the students that this method returns an empty enumeration, if the servlet has no attribute.

Use the code snippet as shown on slide 20 to display the use of the methods in the `FilterConfig` interface. Tell them that in the code, the `init ()` method of a filter is called only once, when the filter is instantiated by the container. The `getInitParameter ()` method retrieves the value passed as parameter to the filter.

Use the figure as shown on slide 21 to display the configuration of filter with initialization parameters in the `web.xml` file.

Tips:

If a definition contains two or more initialization parameters with the same name, then the servlet container will present only one `param-value`, ignoring the others.

In-Class Question:

After you finish explaining initialization parameters, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which method of the filter interface is called by the servlet container to initialize the filter?

Answer:

The `init ()` method of the filter initializes the filter.

Slides 22 and 23

Let us understand use of wildcard in mappings.

Use of Wildcard in Mappings 1-2

- ❖ The `<url-pattern>` element in the deployment descriptor is used to specify URL pattern for the Web resources.
- ❖ It can either contain a specific URL or a even wild card character (*) can be used to match a set of URLs.
- ❖ All the URLs mapped with filters are applied before the Servlet are invoked.
- ❖ When a Web application is executed, it creates an instance of each filter that has been declared in the deployment descriptor.
- ❖ The sequence for execution of these filters is in the order, as they are declared in the deployment descriptor.

© Aptech Ltd.

Web Component Development Using Java/Session 4

22

Use of Wildcard in Mappings 2-2

- ❖ The code snippet shows how to include multiple filters with a request to a mapped Servlet.

```
<filter-mapping>
    <filter-name>LogFilter</filter-name>
    <url-pattern>*.abc</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>AuditFilter</filter-name>
    <url-pattern>*.abc</url-pattern>
</filter-mapping>
...

```

- ❖ The code snippet shows the mapping for multiple patterns with the filter.

```
<filter-mapping>
    <filter-name>DispatchFilter</filter-name>
    <servlet-name>*</servlet-name>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
...

```

© Aptech Ltd.

Web Component Development Using Java/Session 4

23

Use slides 22 and 23 to explain the use of wildcard in mappings.

Tell the students that a wildcard character is a special character that symbolizes one or more characters. The `url-pattern` element of a `filter-mapping` associates a filter with a set of URLs. When a request arrives, the container uses a simple procedure for matching the URL in the request with a `url-pattern` in the `web.xml` file.

Using slide 23, explain the code snippet that demonstrates how to include multiple filters with a request to a mapped Servlet.

Provide them with a situation where a Web page sends a request for a URL, `.../pages/add_customers.abc`, then the filters will be applied to the request in the order: `LogFilter` and then `AuditFilter`.

Servlet API allows to specify multiple match criteria in the same entry. This implies that there can be multiple entries for the `<url-pattern>` element.

You can also have multiple entries for `<servlet-name>` and `<url-pattern>` elements inside the `<filter-mapping>` element.

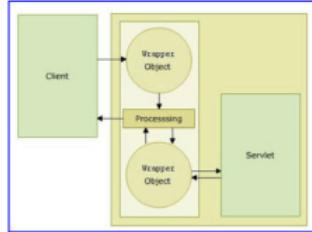
Then, explain the code snippet given on slide 23 to display the mapping for multiple patterns with the filter. Tell them that in this code, the `<filter-mapping>` element defines a filter named DispatchFilter in the `<filter-name>` element to handle all the forward calls to the Servlets. This is achieved using the `<dispatcher>` element having the value set to **FORWARD**.

Slides 24 to 27

Let us understand manipulating requests and responses.

Manipulating Requests and Responses 1-4

- ❖ The filters can wrap the request or response objects in custom wrappers.
- ❖ The wrapper classes:
 - Helps to intercept the request or response, before they can reach their logical destination.
 - Create the object to capture the request and response, before they reach server and client respectively.
- ❖ Figure shows the working of wrapper objects used for processing of request and response in Servlets.



© Aptech Ltd.

Web Component Development Using Java/Session 4

24

Manipulating Requests and Responses 2-4

- ❖ The classes defined by Servlet API for wrapping request and response are as follows:
 - **ServletRequestWrapper**
 - This class provides a convenient implementation of the `ServletRequest` interface.
 - **ServletResponseWrapper**
 - This class provides a convenient implementation of the `ServletResponse` interface.

© Aptech Ltd.

Web Component Development Using Java/Session 4

25

Manipulating Requests and Responses 3-4

- ❖ The code snippet shows how to extend `HttpServletRequestWrapper` class.

```
class MyRequestWrapper extends HttpServletRequestWrapper
{
  ...
  public MyRequestWrapper(HttpServletRequest nested) {
    super(nested);
  ...
}
  public void setParams(String key, String value) {
  ...
}
  public String getParameter(String name) {
    String response;
    if (response==null){
      response=super.getParameter(name);
    }
    return response;
  }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

26

Manipulating Requests and Responses 4-4

- The code snippet shows how to extend `HttpServletResponseWrapper` class.

```
public class MyResponseWrapper extends HttpServletResponseWrapper {
    ...
    public String toString() {
        ...
    }
    public MyResponseWrapper (HttpServletResponse response) {
        super(response);
        ...
    }
    public PrintWriter getWriter() {
        ...
    }
}
```

Use slides 24 to 27 to explain manipulation of requests and responses. Tell the students that a filter can manipulate a request or response in many ways. For example, a filter can add an attribute to the request or can attach data in the response. In addition to perform pre and post processing operations on the request and response objects, the filters can also wrap the request or response objects in custom wrappers.

Use the figure given on slide 24 to display and explain the working of wrapper objects used for processing of request and response in Servlets.

Tell them that the wrapper object generated by the filter implements the `getWriter()` and `getOutputStream()`, which returns a stand-in-stream. The stand-in-stream is passed to the Servlet through the wrapper object. The wrapper object captures the response through the stand-in-stream and sends it back to the filter.

Use slide 25 to explain the classes defined by Servlet API for wrapping request and response as:

- 'ServletRequestWrapper'**

This class provides a convenient implementation of the `ServletRequest` interface. The `ServletRequest` can be subclassed by developers wishing to send the request to a servlet. To override request methods, one should wrap the request in an object that extends `ServletRequestWrapper` or `HttpServletRequestWrapper`.

- 'ServletResponseWrapper'**

This class provides a convenient implementation of the `ServletResponse` interface. The `ServletResponse` can be subclassed by developers wishing to send the response from a servlet. To override response methods, one should wrap the response in an object that extends `ServletResponseWrapper` or `HttpServletResponseWrapper`.

Use the code snippet given on slide 26 to explain how to extend `HttpServletRequestWrapper`.

Then, use the code snippet given on slide 27 to explain how to extend `HttpServletResponseWrapper` class.

Slides 28 and 29

Let us understand annotations.

Introduction to Annotations 1-2

- ❖ Annotations are one of the major advancement from Java EE 5.0 that are used to configure the server components.
- ❖ Using annotation in Java EE, makes the standard `application.xml` and `web.xml` deployment descriptors files optional.

© Aptech Ltd. Web Component Development Using Java/Session 4 28

Introduction to Annotations 2-2

- ❖ **Annotations:**
 - ❑ Can be defined as metadata information that can be attached to an element within the code to characterize it.
 - ❑ Simplifies the developer's work by reducing the amount of code to be written by moving the metadata information into the source code itself.
 - ❑ Can be added to program elements such as classes, methods, fields, parameters, local variables, and packages.
 - ❑ Never executed.
 - ❑ Is processed when the code containing it are compiled or interpreted by compilers, deployment tools, and so on.
 - ❑ Can result in the generation of code documents, code artifacts, and so on.

© Aptech Ltd. Web Component Development Using Java/Session 4 29

Use slides 28 and 29 to explain annotations.

Tell the students that there are two approaches adopted in developing Java server-side components namely, imperative programming and declarative programming. The imperative programming approach specifies how to achieve a goal. The declarative programming specifies only the goal, but not the implementation code to achieve that specific goal.

Prior to Java EE 5, the Java EE platform supported the declarative approach where many of the APIs required code to be written in external files such as the deployment descriptors, server specific configurations, and so on. It would be more convenient if the code to be written in the deployment descriptor is maintained as part of the program itself, but in a declarative way. To avoid writing such kind of unnecessary codes, annotations are used.

Annotations offer data about a program that is not part of the program itself, and they have no direct influence on the operation of the code they annotate. An example of annotation is `@Override`. Here, `@` indicates to the compiler that what comes after it is an annotation, and the annotation's name is `Override`.

Annotations can be used to inform the compiler to identify errors or subdue warnings. Software tools can process annotation information to generate configuration at runtime. Few annotations are accessible to be inspected during compile time and some at runtime.

Slide 30

Let us understand advantages of annotations.

Advantages of Annotations

❖ Some of the advantages of using annotations are as follows:

- Ease of Use**
 - Annotations are checked and compiled by the Java language compiler and are simple to use.
- Portability**
 - Annotations are portable.
- Type Checking**
 - Annotations are instances of annotation types and are compiled in their own class files.
- Runtime Reflection**
 - Annotations are stored in the class files and accessed for runtime access.

© Aptech Ltd. Web Component Development Using Java/Session 4 30

Use slide 30 to list and explain some of the advantages of using annotations.

Slides 31 and 32

Let us understand declaring annotations.

Declaring Annotations 1-2

- ❖ The annotation type is used for defining an annotation and is used when custom annotation needs to be created.
- ❖ An annotation type takes an 'at (@)' sign, followed by the interface keyword and the annotation name.
- ❖ An annotation takes the form of an 'at' (@) sign, followed by the annotation type.
- ❖ The various standard annotations include @Deprecated, @Override, @SuppressWarnings, @Documented, and @Retention.

© Aptech Ltd. Web Component Development Using Java/Session 4 31

Declaring Annotations 2-2

- ❖ The rules and guidelines to be followed by a developer before using annotation are as follows:

A field or a method can be assigned any access qualifier.

A field or method cannot be static.

The fields or methods of the main class that have been annotated for injection must be static.

Resource annotation can be specified in any of the classes or their superclasses.

© Aptech Ltd. Web Component Development Using Java/Session 4 32

Use slides 31 and 32 to explain declaring annotations.

Use slide 32 to list and explain some rules and guidelines to be followed by a developer, before using annotations. Caution them that any violation of these rules will generate an error that will result in logging of message.

Some of the guidelines are as follows:

- A field or a method can be assigned any access qualifier.
- A field or method cannot be static. However, the fields or methods of the main class that have been annotated for injection must be static.
- Resource annotation can be specified in any of the classes or their super classes.
- Any violation of these rules will generate an error that will result in logging of message or messages.

Slide 33

Let us understand support for annotation.

Support for Annotation

- ❖ Servlet API supports different types of annotations to provide information.
- ❖ The `javax.servlet.annotation` package provides annotations to declare Servlets by specifying metadata information in the Servlet class.
- ❖ The `javax.servlet.annotation` package also provides annotations to declare filters and listeners.
- ❖ Figure shows how to configure a Servlet by adding annotations to it.

The diagram illustrates the process of creating a Servlet. It features three main components: a document icon labeled "Servlet Code", a plus sign icon, a circle containing an "@" symbol labeled "Annotation", and an equals sign icon. To the right of these icons is a yellow rounded rectangle labeled "Servlet". This visual metaphor represents how the combination of Java code and annotations results in a fully configured Servlet.

© Aptech Ltd. Web Component Development Using Java/Session 4 33

Use slide 33 to explain support for annotations.

Tell the students that Servlet API supports different types of annotations to provide information. Some annotations can be used as an alternative to XML entries specified in the deployment descriptor, `web.xml`. Other annotations act as a request for the container to perform tasks that would otherwise be performed by the servlet.

Use the figure as shown on slide 33 to demonstrate how to configure a Servlet by adding annotations to it.

Slide 34

Let us understand Annotations API.

Annotations API

- ❖ The javax.servlet.annotation package:
 - Contains a number of annotations Servlets, filters, and listeners.
 - Provides different types of annotations.
 - Figure shows the different types of annotations.

© Aptech Ltd. Web Component Development Using Java/Session 4 34

Use slide 34 to explain annotations API.

Tell the students that if you use annotations to develop the Web application, there is no need to use the deployment descriptor, `web.xml`. You should use higher version of NetBeans IDE and Web servers to run the servlets developed using annotations.

The `javax.servlet.annotation` package contains a number of annotations Servlets, filters, and listeners. Using these packages, the developers can develop a complete Web application in annotation. The `javax.servlet.annotation` package provides different type of annotations.

Using slide 34, list the different types of annotations provided by the `javax.servlet.annotation` package.

In-Class Question:

After you finish explaining annotations API, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which package provides annotations to declare Servlets by specifying metadata information in the Servlet class?

Answer:

The `javax.servlet.annotation` package.

Slides 35 to 39

Let us understand Servlet annotations.

Servlet Annotations 1-5

- ❖ Some of the annotations used by the Servlet are as follows:
- **WebServlet:**
 - This annotation is used to provide the mapping information of the Servlet.
 - Table shows the attributes of @WebServlet annotation.

Attribute Name	Description
name	Specifies the Servlet name. This attribute is optional.
description	Specifies the Servlet description and it is an optional attribute.
displayName	Specifies the Servlet display name, this attribute is optional.
urlPatterns	An array of url patterns use for accessing the Servlet, this attribute is required and should register one url pattern.
asyncSupported	Specifies whether the Servlet supports asynchronous processing or not, the value can be true or false.
initParams	An array of @WebInitParam, that can be used to pass servlet configuration parameters. This attribute is optional.
loadOnStartup	An integer value that indicates servlet initialization order, this attribute is optional.
smallIcon	A small icon image for the servlet, this attribute is optional.
largeIcon	A large icon image for the servlet, this attribute is optional.

© Aptech Ltd.

Web Component Development Using Java/Session 4

35

Servlet Annotations 2-5

- ❖ The code snippet demonstrates how to assign @WebServlet annotation to the Servlet class.

```

    ...
    @WebServlet(
        name = "webServletAnnotation",
        urlPatterns = {"hello", "/helloWebApp"},
        asyncSupported = false
    )
    public class HelloAnnotationServlet extends HttpServlet {
        @Override
        protected void doGet(HttpServletRequest request,
                             HttpServletResponse response) throws ServletException,
                             IOException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            out.write("<html><head><title>WebServlet Annotation</title></head>");
            out.write("<body>");
            out.write("<h1>Servlet Hello Annotation</h1>");
            out.write("<hr/>");
            out.write("Welcome " +
                    getServletConfig().getInitParameter("name"));
            out.write("</body></html>");
            out.close();
        }
    }

```

© Aptech Ltd.

Web Component Development Using Java/Session 4

36

Servlet Annotations 3-5

- ❖ To avoid this wastage of time, there is an option to disable annotation using the metadata-complete attribute in the web.xml file.
- ❖ The code snippet shows how to disable annotation in the Servlet files with version 2.5.

```
...
<web-app version="3.1"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
  http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  metadata-complete="true">
</web-app>
...
```

Servlet Annotations 4-5

❑ **WebInitParam:**

- ❖ This annotation is used to specify the initialization parameters.
- ❖ This can be used with Servlets as well as filters.
- ❖ Table shows the attributes of @WebInitParam annotation.

Attribute Name	Description
name	Specifies the names of the initialization parameters.
value	Specifies the value for the initialization parameters.

Servlet Annotations 5-5

- ❖ The code snippet shows how to apply the @WebInitParam annotation to the servlet.

```
@WebServlet(
    name = "webServletAnnotation", urlPatterns = {"/hello", "/helloWebApp"},
    asyncSupported = false,
    initParams = {
        @WebInitParam(name = "name", value = "admin"),
        @WebInitParam(name = "param1", value = "paramValue1"),
        @WebInitParam(name = "param2", value = "paramValue2")
    }
)
public class HelloAnnotationServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
        out.write("Welcome " + getServletConfig().getInitParameter("name"));
    }
}
```

Use slide 35 to 39 to explain some of the annotations used by the Servlet.

Tell the students that the mapping information of servlet can be provided in servlet file itself, instead of web.xml. The servlet mapping annotations are supported with various elements which are optional.

Use the table as shown on slide 35 to list and explain the attributes of @WebServlet annotation.

Tell them that the @WebServlet annotation is used to provide the mapping information of the Servlet. It is processed by the servlet container at the time of the deployment. The result of processing the annotations provides information about the Servlet to the container such as Servlet class, specified URL pattern to access the Servlet, and so on.

Use the code snippet as shown on slide 36 to demonstrate how to assign @WebServlet annotation to the servlet class. Tell them that if the servlet is already configured in the deployment descriptor, then the deployment descriptor will have higher precedence than annotations. However, this is time consuming since the container has to find all the annotation. In other words, the container has to scan all the files including the jar files present in the classpath.

To avoid this wastage of time, there is an option to disable annotation using the metadatacomplete attribute in the web.xml file. If you set the metadata-complete attribute value to true, then the container will ignore all annotations in the Servlet files. In case, the attribute value is set to false or not specified, then by default, it will search for the annotation declaration.

Use the code snippet as shown on slide 37 to display the code to disable annotation in the Servlet files with version 2.5. Tell them that in this code, the version of Servlet files is specified as 3.1 and the metadata-complete attribute value is set to true, so that the container will disable and ignore all annotations applied in the Servlet files.

Use the table as shown on slide 38 to list and explain the attributes of @WebInitParam annotation.

Use the code snippet as shown on slide 39 to display how to apply the @WebInitParam annotation to the servlet. Tell them that the initParams attribute of @WebServlet is used to pass Servlet configuration parameters.

Slides 40 to 43

Let us understand listener annotations.

Listener Annotations 1-4

- ❖ The Servlet API provides different types of listener interfaces to handle different events.
- ❖ To handle HttpSession life cycle events, the HttpSessionListener can be used.
- ❖ To declare a class as a listener class, the @WebListener annotation can be used.

© Aptech Ltd.

Web Component Development Using Java/Session 4

40

Listener Annotations 2-4

- ❖ The @WebListener annotation is used to register the following types of listeners:

Context Listener	* javax.servlet.ServletContextListener
Context Attribute Listener	* javax.servlet.ServletContextAttributeListener
Servlet Request Listener	* javax.servlet.ServletRequestListener
Servlet Request Attribute Listener	* javax.servlet.ServletRequestAttributeListener
Http Session Listener	* javax.servlet.http.HttpSessionListener
Http Session Attribute Listener	* javax.servlet.http.HttpSessionAttributeListener

© Aptech Ltd.

Web Component Development Using Java/Session 4

41

Listener Annotations 3-4

- ❖ The code snippet shows the implementation of the class that will log the session created or destroyed.

```

. . .
import javax.servlet.annotation.WebListener;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;
@WebListener
public class MySessionListener implements HttpSessionListener{
    @Override
    public void sessionCreated(HttpSessionEvent se) {
        System.out.println("Session Created:: ID="+ se.getSession().
        getId());
    }
    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        System.out.println("Session Destroyed:: ID="+ se.getSession().
        getId());
    }
}

```

© Aptech Ltd.

Web Component Development Using Java/Session 4

42

Listener Annotations 4-4

- The code snippet shows the Servlet class to create the HttpSession object.

```
@WebServlet("/MyServlet")
public class MyServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response) throws ServletException,
                         IOException
    {
        HttpSession session = request.getSession();
        session.invalidate();
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

43

Use slides 40 to 43 to explain listener annotations.

Use slide 40 to explain about use of listeners in servlets.

Use slide 41 to list the types of listeners that are registered using @WebListener annotation. Use the code snippet as shown on slide 42 to demonstrate the implementation of the class that will log whenever the session is created or destroyed.

Use the code snippet as shown on slide 43 to explain the Servlet class to create the HttpSession object. Tell them that the code creates an HttpSession object to store the session object. The session object is also destroyed to check the invocation of listeners.

Slides 44 and 45

Let us understand filter annotations.

Filter Annotations 1-2

- ❖ In a Web application, the `@WebFilter` annotation is used to define a filter.
- ❖ It is specified on a class and contains metadata regarding the filter being declared.
- ❖ At least one URL pattern must be specified with the annotated filter.
- ❖ The `urlPatterns` or `value` attribute on the annotation accomplishes this.
- ❖ The other attributes are optional and have default settings.

© Aptech Ltd.

Web Component Development Using Java/Session 4

44

Filter Annotations 2-2

- ❖ The code snippet shows the creation of filter using `@WebFilter` annotation.

```
@WebFilter (value="/hello",
           initParams={@WebInitParam(name="message", value="Servlet
says: ")})
public MyFilter implements Filter {
    private FilterConfig _filterConfig;
    public void init(FilterConfig filterConfig)
        throws ServletException
    {
        filterConfig = filterConfig;
    }
    public void doFilter(ServletRequest req,
                        ServletResponse res,
                        FilterChain chain)
        throws ServletException, IOException
    {
        PrintWriter out = res.getWriter();
        out.print(_filterConfig.getInitParameter("message"));
    }
    public void destroy() {
        // destroy
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

45

Use slides 44 and 45 to explain filter annotations.

Tell the students that `value` attribute can be used when the annotation is only provided with the URL pattern. The `urlPatterns` attribute can be used when other attributes are also used. In addition, classes annotated with the `@WebFilter` annotation must implement the `javax.servlet.Filter` interface.

To add configuration data to the filter, you should specify the `initParams` attribute of the `@WebFilter` annotation. The `initParams` attribute contains a `@WebInitParam` annotation.

Use the code snippet as shown on slide 45 to explain the creation of filter using `@WebFilter` annotation.

Slide 46

Let us understand security annotations.

Security Annotations

- ❖ Security annotations can be used to specify permission on the methods of the Servlet class.
- ❖ Some of the security annotations are as follows:

ServletSecurity	It is used on a Servlet class to specify security constraints to be enforced by a servlet container on HTTP protocol messages.
HttpConstraint	It is used within the <code>ServletSecurity</code> annotation to represent the security constraints to be applied to all HTTP protocol methods.
HttpMethodConstraint	It is used within the <code>ServletSecurity</code> annotation to represent security constraints on specific HTTP protocol messages.

© Aptech Ltd. Web Component Development Using Java/Session 4 46

Use slide 46 to list and explain security annotations.

Some of the security annotations are as follows:

ServletSecurity - This annotation is used on a Servlet class to specify security constraints to be enforced by a servlet container on HTTP protocol messages.

HttpConstraint - This annotation is used within the `ServletSecurity` annotation to represent the security constraints to be applied to all HTTP protocol methods.

For example, to restrict the user from accessing any other method in the Servlet class, when logged in the role as 'User', then the annotation will be:

```
@ServletSecurity(@HttpConstraint(rolesAllowed = "User"))
```

HttpMethodConstraint - This annotation is used within the `ServletSecurity` annotation to represent security constraints on specific HTTP protocol messages.

Provide them with an example for `HttpMethodConstraint` annotation as:

```
@ServletSecurity(@HttpMethodConstraint(value = "POST"), emptyRoleSemantic = EmptyRoleSemantic.DENY).
```

Slides 47 and 48

Let us understand dependency injection annotations.

Dependency Injection Annotations 1-2

- ❖ Dependency Injection (DI) mechanism allows the container to inject the dependent objects in the application components.
- ❖ The application components can be dependent on many resources to perform the necessary operations.
- ❖ The dependent objects are injected before the life cycle methods are invoked and before any reference is made to the dependent object.
- ❖ Servlet API supports various dependency injection annotations for different types of resources on which Servlet can be dependent. These include:
 - @Resource
 - @RJB
 - @PersistenceUnit
 - @PersistenceContext
 - @WebServiceRef

© Aptech Ltd.

Web Component Development Using Java/Session 4

47

Dependency Injection Annotations 2-2

- ❖ The code snippet demonstrates how to obtain a reference for the data source in the Servlet class.

```
// Injecting the Datasource named EmployeeDB
@Resource(name="jdbc/EmployeeDB")
private javax.jdbc.DataSource myDB;
// Creating connection with the data source
Connection con;
// obtaining the connection
con = myDb.getConnection();
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

48

Use slides 47 and 48 to explain dependency injection annotations.

Tell the students that the application components can be dependent on many resources to perform the necessary operations. For example, the requirement of an datasource object to connect to different components such as a database, a JavaBean object, an enterprise bean object, and so on.

Dependency injection enables the developers to change regular Java classes into managed objects and to inject them into any other managed object. You can utilize dependency injection to enable the code to declare dependencies on any managed object. The container provides instances of the dependencies at the injection points at runtime, and it also manages the life cycle of the instances injected in the code.

Use the code snippet as shown on slide 48 to demonstrate the code that can be used to obtain a reference for the data source in the Servlet class. Tell them that the @Resource annotation is used for accessing data source registered in JNDI service on the server.

Slide 49

Let us understand uploading files with Servlet.

The slide has a blue header bar with the title 'Uploading Files with Servlet'. Below the header, there are two bullet points:

- ❖ A very common requirement of a Web application is to upload the files on the server.
- ❖ File can be uploaded on the sever using the following two ways:

Below these points are two numbered options:

1. Client-side file upload
2. Server-side file upload

At the bottom of the slide, there is footer text: '© Aptech Ltd.', 'Web Component Development Using Java/Session 4', and '49'.

Use slide 49 to explain uploading files on the server and list the two ways to do it.

Tell the students that one of the basic and common requirement of a Web application is enabling easy file uploads. Earlier, this task required the help of complicated input processes and external libraries. Now, Java Servlet specification offers a feasible method of file uploads out of the box. Any Web container that implements the specification can parse multipart requests and make MIME attachments available through the `HttpServletRequest` object.

A very common requirement of a Web application is to upload the files on the server. File can be uploaded on the sever using the following two ways:

- Client-side file upload
- Server-side file upload

Slides 50 to 52

Let us understand client-side uploading.

Client-side Uploading 1-3

- ❖ To upload the files on Web pages, there are some changes required in the HTML form.
- ❖ The requirements are as follows:
 - ❑ Create the input element with the attribute `type="file"`.
 - ❑ The form method supported for file upload will be POST, as GET method is not supported in file uploading.
 - ❑ Use the attribute named `enctype` with the `form` element.
- ❖ Table shows the values that can be assigned to the `enctype` attribute.

Value	Description
<code>application/x-www-form-urlencoded</code>	<ul style="list-style-type: none"> • This is the default value if not specified. • The value ensures that all characters are encoded before they are sent to the server.
<code>multipart/form-data</code>	<ul style="list-style-type: none"> • The value is provided when the form is using a file upload control. • It does not encode the characters before sending them in the request.
<code>text/plain</code>	<ul style="list-style-type: none"> • The value converts the spaces with the "+" symbols, however no other special characters are encoded.

© Aptech Ltd.

Web Component Development Using Java/Session 4

50

Client-side Uploading 2-3

- ❖ The code snippet shows the `upload.jsp` page containing the HTML form element to upload the file on the server.

```

<html>
<head>
<title>File Uploading Form</title>
</head>
<body>
<h3>File Upload:</h3>
Select a file to upload: <br />
<form action="UploadServlet" method="post"
      enctype="multipart/form-data">
<input type="file" name="file" size="50"/>
<br />
<br />
<input type="submit" value="Upload File" />
</form>
</body>
</html>

```

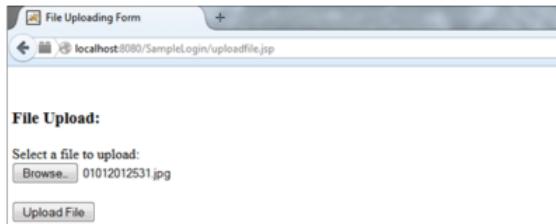
© Aptech Ltd.

Web Component Development Using Java/Session 4

51

Client-side Uploading 3-3

- ❖ Figure shows the output of the JSP page.



© Aptech Ltd.

Web Component Development Using Java/Session 4

52

Use slides 50 to 52 to explain client-side uploading of files on the server.

To upload the files on Web pages, there are some changes required in the HTML form.

The requirements are as follows:

- Create the input element with the attribute `type="file"`.
- The form method supported for file upload will be `POST`, as `GET` method is not supported in file uploading.
- Use the attribute named `enctype` with the `form` element. This element specifies how the form data will be encoded when the form is submitted for processing on the server.

Use the table given on slide 50 to explain the values that can be assigned to the `enctype` attribute.

Use the code snippet as shown on slide 51 to explain the `upload.jsp` page containing the HTML form element to upload the file on the server. Tell them that the `file` element displays the **Browse** button on the form which allows the user to select the file from the system. When the user clicks `Upload File`, the HTTP request will pass the uploaded file to the servlet, which will write the uploaded file to the folder on the server.

Use the figure as shown on slide 52 to show the output of the JSP page.

Slides 53 to 56

Let us understand server-side file upload.

Server-side File Upload 1-4

- ❖ The `HttpServletRequest` class includes two methods for handling the file upload. These are as follows:
 - `Part getPart(String name)`
 - This method extracts the individual parts of the file and returns each as a `Part` object.
 - The `Part` object belongs to the `Part` interface and provide methods that can be used to extract information from the returned `Part` object.
 - The information includes the name, the size, and the content-type of the file returned as a part.
 - It also provide methods for querying the header content submitted with the `Http` request object, writing the part to the external disk, and deleting the part.
 - `Collection<Part> getParts()`
 - This method returns a collection of `Part` objects which can be iterated to extract individual `Part` object from the collection.

© Aptech Ltd.

Web Component Development Using Java/Session 4

53

Server-side File Upload 2-4

- ❖ The code snippet demonstrates the Servlet code that uploads the file on the server.

```
public class UploadServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
                      HttpServletResponse response) throws IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        boolean isMultipartContent = ServletFileUpload.isMultipartContent(request);
        if (!isMultipartContent) {
            out.println("No file uploaded<br/>");
            return;
        }
        out.println("You are trying to upload:<br/>");
        FileItemFactory factory = new DiskFileItemFactory();
        ServletFileUpload upload = new ServletFileUpload(factory);
        try {
            List<FileItem> fields = upload.parseRequest(request);
            out.println("Number of fields: " + fields.size() + "<br/><br/>");
            Iterator<FileItem> it = fields.iterator();
            if (!it.hasNext()) {
                out.println("No fields found");
            }
        }
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

54

Server-side File Upload 3-4

```
while (it.hasNext()) {
    FileItem fileItem = (FileItem) it.next();
    boolean isFormField = fileItem.isFormField();
    if (!isFormField) {
        String fileName = fileItem.getName();
        File file = new File(fileName);
        fileItem.write(file);
        out.println("Uploaded filename: " + fileName +
                    "<br>");
    }
}
catch (Exception e) {
    e.printStackTrace();
}
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

55

Server-side File Upload 4-4

- The code snippet shows the corresponding `web.xml` file for configuring the Servlet.

```
<servlet>
    <servlet-name>UploadServlet</servlet-name>
    <servlet-class>com.Upload.UploadServlet</servlet-
class>
</servlet>

<servlet-mapping>
    <servlet-name>UploadServlet</servlet-name>
    <url-pattern>/UploadServlet</url-pattern>
</servlet-mapping>

<welcome-file-list>
    <welcome-file>uploadfile.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

Use slides 53 to 56 to explain server-side file upload.

Tell the students that in server-side file upload, the files are uploaded using servlet class. The servlet specification provides the classes that are used to handle encryption from data. The classes that are involved in handling the file uploading in the Servlet are `HttpServletRequest`.

Use the code snippet as shown on slides 54 and 55 to demonstrate the servlet code that uploads the file on the server.

Use the code snippet given on slide 56 to display the corresponding `web.xml` file for configuring the servlet.

Slides 57 to 59

Let us understand MultipartConfig annotation.

MultipartConfig Annotation 1-3

- ❖ The `javax.servlet.annotation.MultipartConfig` annotation is specified on a Servlet class.
- ❖ It provides file information to the server while uploading its content.
- ❖ The information provided by the `MultipartConfig` annotation is as follows:

The maximum size of the data that can be uploaded on the server

The default location where the file will be uploaded on the server

© Aptech Ltd.

Web Component Development Using Java/Session 4

57

MultipartConfig Annotation 2-3

- ❖ Table lists the attributes of the `MultipartConfig` annotation.

Attribute	Description
<code>fileSizeThreshold</code>	• Specifies the size of the file in bytes.
<code>maxFileSize</code>	• Specifies the maximum file limit allowed to upload the file on the server.
<code>maxRequestSize</code>	• Specifies the maximum file size in bytes to be accepted for encrypted file data sent in the request.
<code>location</code>	• Specifies the default location for the file upload on the server.

© Aptech Ltd.

Web Component Development Using Java/Session 4

58

MultipartConfig Annotation 3-3

- ❖ The code snippet shows the attributes of `@MultipartConfig` annotation that are applied to the Servlet.

```
@WebServlet(name = "UploadServlet",  
urlPatterns = {"/*Upload"})  
@MultipartConfig(  
    fileSizeThreshold=100,  
    maxFileSize=400,  
    maxRequestSize=50,  
    location="C:/tmp")  
  
public class UploadServlet extends HttpServlet {  
    protected void doPost(HttpServletRequest request,  
    HttpServletResponse response) throws IOException {  
        . . .  
    }  
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 4

59

Use slides 57 to 59 to explain MultipartConfig annotation. It provides file information to the server while uploading its content.

Use the table as shown on slide 58 to list and explain the attributes of the `MultipartConfig` annotation.

While explaining `fileSizeThreshold` attribute, mention to the students that the default size of the file is 0. After explaining `maxFileSize` attribute, tell them that if the size of the file to be uploaded is greater than the specified limit, then `IllegalStateException` is thrown.

Use the code snippet given on slide 59 to display the attributes of `@multipartconfig` annotation that are applied to the Servlet.

Slide 60

Let us summarize the session.

Summary

- ❖ Filter acts as an interface between client and servlet inside the server. It intercepts the request or response to and from the server. More than one filter can also exist between a client and a servlet. This is called a filter chain.
- ❖ The Filter API is a part of the servlet package. It contains interfaces namely, Filter, FilterConfig, and FilterChain.
- ❖ All the information about the filter is described within the <filter> element inside the web.xml file. The filter also needs to be mapped to the servlet for which it will function. This is done by assigning URL name and servlet name inside the <filter-mapping> element.
- ❖ The request and response to and from the servlet is manipulated by the filter. The wrapper object of RequestWrapper or ResponseWrapper classes created by the filter intercepts the request or response and sends it to the filter.
- ❖ Annotation can be defined as metadata information that can be attached to an element within the code to characterize it. Annotation can be added to program elements such as classes, methods, fields, parameters, local variables, and packages.
- ❖ The javax.servlet.annotation package contains a number of annotations Servlets, filters and listeners. Apart from this, Servlet API also supports various dependency injection annotations.
- ❖ File can be uploaded on the server using the following two ways namely, Client-side file upload and server-side file upload.
- ❖ The javax.servlet.annotation.MultipartConfig annotation is specified on a Servlet class to provide file information to the server while uploading its content.

© Aptech Ltd. Web Component Development Using Java/Session 4 60

In slide 60, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

4.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore session handling, session events, and listener interfaces used in servlets.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 5 – Database Access and Event Handling

5.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

5.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain database handling using JDBC
- Describe JDBC and its role
- Explain connecting database using JDBC
- Describe JPA and its role
- Describe connecting database using JPA
- Explain the significance of session handling and session events
- Explain different types of listener interfaces used in Servlets

5.1.2 Teaching Skills

To teach this session successfully, you should have thorough knowledge about the concept of establishing the database connection in the servlet using the Java Database Connectivity (JDBC) API. Also, familiarize yourself with the concept of Object Relational Mapping (ORM) and persisting the entities in the database using Java Persistence (JPA) API.

The session also covers event-handling mechanism in servlets.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives
<ul style="list-style-type: none">❖ Explain database handling using JDBC❖ Describe JDBC and its role❖ Explain connecting database using JDBC❖ Describe JPA and its role❖ Describe connecting database using JPA❖ Explain the significance of session handling and session events❖ Explain different types of listener interfaces used in Servlets

© Aptech Ltd.

Web Component Development Using Java/Session 5

2

Tell them that they will be introduced to various mechanisms, using which data can be accessed from the database. JDBC API provides classes and interfaces that allow a Web application to access and perform operations on the databases. The session will also teach an ORM technology named JPA that is used to persist entities from a Web application into the databases.

Then, the session explains the significance of event handling in the servlets. It explains different types of listeners used in servlets.

5.2 In-Class Explanations

Slides 3 and 4

Let us understand the mechanisms used in Java for persistence.

Introduction 1-2

Java provides various mechanisms using which data can be accessed from the database. These are as follows:

- ❖ **Using Java Database Connectivity (JDBC) API**
 - It is a part of Java SE platform that enables a Web application to interact with a Database Management System (DBMS).

© Aptech Ltd. Web Component Development Using Java/Session 5 3

Introduction 2-2

- ❖ **Using Object Relational Mapping (ORM) API**
 - It is a persistence mechanism used by enterprise application to access relational databases.
 - It maps the Java objects to database tables through XML configuration.

© Aptech Ltd. Web Component Development Using Java/Session 5 4

Use slides 3 and 4 to explain various mechanisms provided by Java to access data from the databases.

Tell the students that the most common operations performed by a Servlet include storing and retrieving data from a database.

Tell them that JDBC API is a connectivity mechanism between the Java programming language and a wide range of databases. It is a part of Java SE platform that enables a Web application to interact with a Database Management System (DBMS).

Using the figure shown on slide 3, explain the use of JDBC API with respect to Java applications.

Explain them about Object Relational Mapping (ORM). It is a persistence mechanism used by enterprise application to access relational databases. It maps the Java objects to database tables through XML configuration. The Java Persistence API (JPA) is one such specification that is based on ORM. Then using the figure shown on slide 4, explain the mechanism of ORM API.

Tips:

A relational database presents information as a collection of tables. A table provides a structure in the form of rows and columns to store and retrieve data from the database. To learn more on database concepts for the session, you can refer this link:

<http://docs.oracle.com/javase/tutorial/jdbc/overview/database.html>

Slide 5

Let us understand JDBC API.

JDBC API

- ❖ Provides classes and interfaces that allow a Web application to access and perform operations on the databases.
- ❖ The operations that can be performed on the databases include:
 - ❑ Connecting to the databases
 - ❑ Processing SQL statements
 - ❑ Building result sets
 - ❑ Executing the parameterized statements
 - ❑ Invoking callable statements

© Aptech Ltd. Web Component Development Using Java/Session 5 5

Use slide 5 to explain and list the operations that can be performed on the database using JDBC API.

Tell them that JDBC API provides classes and interfaces that allow a Web application to access and perform operations on the databases.

Then, list the type of operations that can be performed by JDBC API that are as follows:

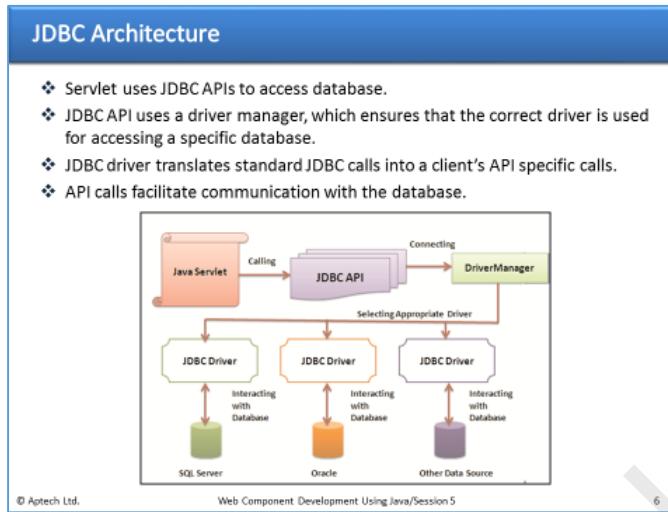
- Connecting to the databases
- Processing SQL statements
- Building result sets
- Executing the parameterized statements
- Invoking callable statements

Tips:

The JDBC API supports both two-tier and three-tier processing models for database access. With increase in the development of Java-based enterprise applications, JDBC API is used mainly on middle-tier of a three-tier architecture. Some of the features that make JDBC useful on the middle-tier include connection pooling, distributed transactions, and disconnected rowsets.

Slide 6

Let us understand JDBC architecture.



Use slide 6 to describe JDBC architecture.

Use the figure shown on slide 6 to explain the JDBC architecture. The servlet code uses JDBC APIs to access database. The JDBC API uses a driver manager, which ensures that the correct driver is used for accessing a specific database. A **JDBC driver** is a software component enabling a Java application to interact with a database.

A JDBC driver translates standard JDBC calls into a client's API specific calls. These API calls facilitate communication with the database. Thus, the developer working with JDBC API need not bother about repackaging the application, if the database is changed or upgraded.

Some of the classes and interfaces provided by JDBC API are as follows:

- Driver Interface – handles the communications with the database server.
- DriverManager Class – manages database drivers and links connection requests from the application to the correct database driver.
- ResultSet Objects – stores data recovered from a database once SQL query is implemented using Statement objects.
- SQL Exception Class – takes care of any errors that might crop up in a database application.
- Statement Objects – used to submit SQL statement to the database.
- Connection Interface – interfaces with all methods for linking to a database.

Tips:

JDBC requires the database drivers for each database to connect with them individually. The JDBC driver gives out the connection to the database and implements protocol for transferring the query as well as query result between client and database.

Slides 7 to 10

Let us understand JDBC driver types.

JDBC Driver Types 1-4

- ❖ **Type 1: JDBC-ODBC Bridge Plus ODBC Driver**
 - Provides access to the database through standard ODBC libraries.
 - Translates the standard JDBC calls to the corresponding ODBC calls.
 - Part of the sun.jdbc.odbc package.
 - Capabilities depend on the capabilities of the ODBC driver.
 - Figure shows the JDBC-ODBC bridge driver.

The diagram illustrates the JDBC-ODBC Bridge architecture. On the left, labeled 'Client', there is a stack of three green boxes: 'Application' at the top, 'JDBC Driver' in the middle, and 'ODBC Driver' at the bottom. On the right, labeled 'Database', is a stack of three orange cylinders representing the database. A double-headed blue arrow connects the JDBC Driver and the ODBC Driver. Another double-headed blue arrow labeled 'SQL Request/Results' connects the ODBC Driver to the Database. Arrows also point from the Application down to the JDBC Driver, and from the JDBC Driver down to the ODBC Driver.

© Aptech Ltd.

Web Component Development Using Java/Session 5

7

JDBC Driver Types 2-4

- ❖ **Type 2: Native API Partly Java Driver**
 - Calls the native database library that accesses the database.
 - Requires the native database libraries to be installed and configured on the client's machine.
 - Figure shows native API partly Java driver.

The diagram illustrates the Native API Partly Java Driver architecture. On the left, labeled 'Client', there is a stack of three green boxes: 'Application' at the top, 'JDBC Driver' in the middle, and 'CLI (Call Level Interface) Native Database Libraries' at the bottom. On the right, labeled 'Database', is a stack of three orange cylinders representing the database. A double-headed blue arrow connects the JDBC Driver and the Native Database Libraries. Another double-headed blue arrow labeled 'SQL Request/Results' connects the Native Database Libraries to the Database. Arrows also point from the Application down to the JDBC Driver, and from the JDBC Driver down to the Native Database Libraries.

© Aptech Ltd.

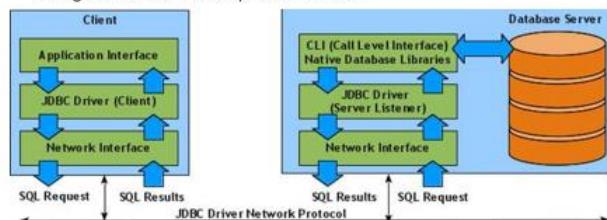
Web Component Development Using Java/Session 5

8

JDBC Driver Types 3-4

❖ **Type 3: JDBC-Net Pure Java Driver**

- Communicates with the JDBC middle tier on the server through a proprietary network protocol.
- Java Servlet sends a JDBC call to the middleware through the JDBC driver without translation, middleware then sends the request to the database.
- Figure shows JDBC-Net pure Java driver.



© Aptech Ltd.

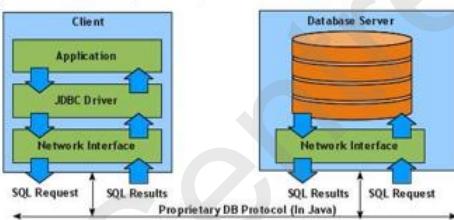
Web Component Development Using Java/Session 5

9

JDBC Driver Types 4-4

❖ **Type 4: Native-Protocol Pure Java Driver**

- Fastest driver.
- Implements a proprietary database driver.
- Provides database connectivity through the standard JDBC APIs and do not require any native libraries on the client machine.
- Figure shows native-protocol pure Java driver.



© Aptech Ltd.

Web Component Development Using Java/Session 5

10

Use slides 7 to 10 to explain JDBC driver types.

Tell them that Java supports four types of drivers to connect and interact with the databases. These are as follows:

- Type 1: JDBC-ODBC Bridge Plus ODBC Driver
- Type 2: Native API Partly Java Driver
- Type 3: JDBC-Net Pure Java Driver
- Type 4: Native-Protocol Pure Java Driver

➤ **Type 1: JDBC-ODBC Bridge Plus ODBC Driver** – The JDBC-ODBC bridge provides access to the database through standard ODBC libraries. The driver translates the standard JDBC calls to the corresponding ODBC calls. This driver is a part of the `sun.jdbc.odbc` package. Its major limitation is that the JDBC driver's capabilities depend on the capabilities of the ODBC driver.

Use the figure shown on slide 7 to display and explain JDBC-ODBC Bridge plus ODBC driver. Tell them that the advantage of this driver is that it can access and recover data of almost any database for which an ODBC driver is installed.

- **Type 2: Native API Partly Java Driver** - The native APIs partly Java driver calls the native database library that accesses the database. This driver also requires the native database libraries to be installed and configured on the client's machine.

Use the figure shown on slide 8 to display and explain Native API Partly Java Driver. Tell them that the advantage of this driver is that it is significantly faster than type 1 driver, as there is no implementation of JDBC-ODBC bridge. However, the type 2 drivers are platform dependent. For example, Oracle Call Interface (OCI) driver.

- **Type 3: JDBC-Net Pure Java Driver** – This is a pure Java driver that makes use of a middle-tier for making the calls between the Java program and the database. The middle-tier (application server) converts JDBC calls directly or indirectly into the vendor-specific database protocol. The Java Servlet sends a JDBC call to the middleware through the JDBC driver without translation. The middleware then sends the request to the database.

Use the figure given on slide 9 to display and explain JDBC-Net Pure Java Driver. Tell them that this driver is exceptionally flexible, as there is no need for code to be installed on the client and a single driver can essentially offer access to numerous databases.

- **Type 4: Native-Protocol Pure Java Driver** – The type 4 drivers implement a proprietary database driver. It provides database connectivity through the standard JDBC APIs and do not require any native libraries on the client machine.

Use the figure shown on slide 10 to display and explain Native-Protocol Pure Java Driver. Tell them that this driver is also exceptionally flexible, as there is no need to install special software on the client or server. In addition, these drivers can be downloaded dynamically. For example, MySQL's Connector/J driver.

In-Class Question:

After you finish explaining various types of JDBC drivers, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which is the fastest JDBC driver?

Answer:

Native-Protocol Pure Java Driver

Tips:

The type of driver to be used depends on the requirement of the applications.

If you want to access one kind of database, for example, IBM, Sybase, or Oracle, the ideal driver type is 4. Type 3 is the ideal driver, if Java application accesses multiple types of databases at the same time. Type 1 driver is generally used only for development and testing purposes.

Slide 11

Let us understand accessing database using JDBC.

Accessing Database Using JDBC

- ❖ Servlet interacts with the database using JDBC APIs and follows certain steps in sequence.

1. Registering the JDBC driver
2. Establishing a database connection
3. Executing an SQL statement
4. Processing the results
5. Closing the database connection

© Aptech Ltd. Web Component Development Using Java/Session 5 11

Use slide 11 to list the sequence of steps to be followed by the servlet to access database using JDBC.

Slide 12

Let us understand registering the JDBC driver.

Registering the JDBC Driver

- ❖ Notifies the driver manager that a JDBC driver is available.
- ❖ JDBC automatically registers itself with the driver manager as soon as it is loaded.
- ❖ JDBC driver can be loaded at run time using the `forName()` method.
- ❖ **Syntax:**

```
Class.forName(String driver-name)
```

- ❖ The `forName()` is a static method that instructs the JVM to locate, load, and link the specified driver classes.

© Aptech Ltd. Web Component Development Using Java/Session 5 12

Use slide 12 to explain how to register the JDBC driver.

Registering the driver notifies the driver manager that a JDBC driver is available. The JDBC automatically registers itself with the driver manager as soon as it is loaded.

Using slide 12, explain the syntax of `forName()` method. Tell them that the `forName()` is a static method that instructs the JVM to locate, load, and link the specified driver classes.

Slides 13 and 14

Let us understand establishing a database connection.

Establishing a Database Connection 1-2

- ❖ Identified by a database URL.
- ❖ Established using the `getConnection()` method of the `DriverManager` class in JDBC API.
- ❖ Syntax for specifying the URL:
`jdbc:<subprotocol>:<database_identifier>`
- ❖ where,
 - JDBC is used for establishing the connection.
 - `<subprotocol>` is the name of any valid database driver or any other database connectivity that is being requested.
 - `<database_identifier>` is the logical name of the database connection that maps to the physical database.

© Aptech Ltd. Web Component Development Using Java/Session 5 13

Establishing a Database Connection 2-2

- ❖ The code snippet demonstrates the connection to a SQL Server database from the Servlet.

```
public class JDBCServlet implements HttpServlet {
    Connection conn = null;
    try {
        // Loads Type 4 driver for connecting to SQL Server
        Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
        // obtains a connection to the database
        conn = DriverManager.getConnection("jdbc:sqlserver://localhost:1433;" +
            "database=ProductStore;user=sa;password=sa;");
    } catch (ClassNotFoundException ex) {
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 5 14

Use slides 13 and 14 to explain establishing a database connection.

The database connection can be established when a valid driver is loaded. The JDBC connection is identified by a database URL.

Using slide 13, display and explain the syntax for specifying the URL. Mention that the `<database_identifier>` is basically the Data Source Name (DSN) that holds the information to establish a connection.

Using the code snippet as shown on slide 14, explain how to establish a connection to a SQL Server database from the servlet. Tell them that, in the code, the `Conn` object is created using the `getConnection()` method of the `DriverManager` class.

Slides 15 and 16

Let us understand executing an SQL statement.

Executing an SQL Statement 1-2

- ❖ To query the database, the `Statement` interface of JDBC API can be used.
- ❖ Figure shows the `Statement` object.

The diagram illustrates the relationship between a `Connection` and a `Statement`. A `Connection` object contains a `Statement Pool`, which holds multiple `Statement` objects (labeled `S0`, `S1`, `S2`, etc.). One `Statement` object is shown expanded to reveal its methods: `Sql query` and `Result Set`.

- ❖ The `Statement` object reference can be created using the `createStatement()` method on the obtained connection.
- ❖ For example, `Statement stat = conn.createStatement();` is used to create the `Statement` object.

© Aptech Ltd. Web Component Development Using Java/Session 5 15

Executing an SQL Statement 2-2

- ❖ The `executeQuery()` method of `Statement` interface accepts SQL query as argument and returns a `ResultSet` object.
- ❖ The `ResultSet` object contains all the rows returned by the SQL query.
- ❖ For example, `ResultSet rs = stat.executeQuery("SELECT * FROM Product");` shows the creation of query using `Statement` object.
- ❖ Figure shows the return of the `ResultSet` object on query execution.

The diagram illustrates the process of executing an SQL query. It shows a `CLIENT` sending an `SQL` query to a `SERVER`. The `SERVER` processes the query and returns a `Resultset` object to the `CLIENT`. A cylinder labeled `DATABASE` is shown near the `SERVER`, indicating its role in storing and retrieving data.

© Aptech Ltd. Web Component Development Using Java/Session 5 16

Use slides 15 and 16 to explain executing an SQL statement.

Tell the students that once the database connection is established, several SQL statements such as retrieving, inserting, updating, or deleting of data can be performed on the database.

Use the figure shown on slide 15 to display and explain the `Statement` object. Tell them that the `Statement` interface provides methods to perform various database operations including SQL query execution. The `Statement` object reference can be created using the `createStatement()` method on the obtained connection. For example, `Statement stat = conn.createStatement();`

The `Statement` interface provides methods to perform various database operations including SQL query execution.

The `Statement` object executes a SQL statement using the following methods:

- **boolean execute(String SQL):** Returns a boolean value of true, if a `ResultSet` object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
- **int executeUpdate(String SQL):** Returns the numbers of rows affected by the execution of the SQL statement. This method is used to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
- **ResultSet executeQuery(String SQL):** Returns a `ResultSet` object. Use this method when you expect to get a result set, as you would with a SELECT statement.

Use the figure shown on slide 15 to display and explain the return of the `ResultSet` object on query execution.

Using slide 16, explain the `ResultSet` object that contains all the rows returned by the SQL query. Mention that in case of any error, the `executeQuery()` method throws an exception – `SQLException`.

Slides 17 and 18

Let us understand processing the ResultSet.

Processing the ResultSet 1-2

- ❖ ResultSet object maintains a cursor which points to the current row in the result table.
- ❖ Three types of ResultSet are as follows:
 - TYPE_FORWARD_ONLY – cursor moves only in forward.
 - TYPE_SCROLL_INSENSITIVE – cursor scrolls forward and backward; not sensitive to changes, which have been made by other users to the database.
 - TYPE_SCROLL_SENSITIVE – cursor scrolls forward and backward; sensitive to changes, which have been made by other users to the database.

© Aptech Ltd.

Web Component Development Using Java/Session 5

17

Processing the ResultSet 2-2

- ❖ The code snippet demonstrates how to iterate through the resultset.


```
// Access column values
while(rs.next()) {
    rs.getInt("product_id");
    rs.getString("product_name");
}
```
- Iterating the ResultSet includes next () method which moves the cursor forward one row.
- Returns true, if the cursor is now positioned on a row. Otherwise false, if the cursor is positioned after the last row.
- ❖ The code snippet demonstrates how to update the resultset.


```
rs.updateString("product_name", "Beverages");
rs.updateRow();
```
- ❖ The statement, conn.close () closes the connection.

© Aptech Ltd.

Web Component Development Using Java/Session 5

18

Use slides 17 and 18 to explain processing the ResultSet.

Tell the students that ResultSet is an interface used to describe ResultSet object. This object represents data in a tabular form that is retrieved by executing an SQL query. Mention that ResultSet follows the iterative pattern.

The code snippet as shown on slide 18 demonstrates the code that iterates through the resultset. Tell them that iterating the ResultSet includes next () method which moves the cursor forward one row. This returns true, if the cursor is now positioned on a row and false, if the cursor is positioned after the last row.

Then, explain the next code snippet given on slide 18 to demonstrate the code that is used to update the resultset.

Explain the last step for closing the database connection. The statement, conn.close () closes the connection explicitly that allows the garbage collector to recollect memory as early as possible. Tell them that it is particularly important to close database-related connections. In case we do not

close them, there will be unclosed connections to the database and it could eventually run out of connections.

In-Class Question:

After you finish explaining the steps to access JDBC, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which method is used to load JDBC driver at runtime?

Answer:

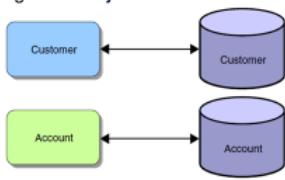
The `forName()` method.

Slide 19

Let us understand Java Persistence API (JPA).

Java Persistence API (JPA)

- ❖ **JPA:**
 - ❑ Lightweight, POJO-based Java framework.
 - ❑ Helps to persist the Java objects to the relational database.
 - ❑ Provides the persistence providers in the JPA application.
 - ❑ Converts data between incompatible type in object-oriented programming to the appropriate underlying database types.
- ❖ ORM technology that persist the entities in the database, supports the mapping to object-oriented software to the database schema through configuration mapping.
- ❖ Figure shows mapping of **Java Objects to Tables**.



© Aptech Ltd. Web Component Development Using Java/Session 5 19

Use slide 19 to explain Java Persistence API (JPA).

Tell the students that JPA provides a POJO persistence model for object-relational mapping of entities in the database. The main benefit of using an ORM technology is that it supports the mapping of object-oriented software to the database schema through configuration mapping. The configuration mapping provides information about the entity in the application domain and its mapping to the database table. The configuration mapping can be specified either in XML file or through annotations.

Tell them that JPA converts data between incompatible types in object-oriented programming to the appropriate underlying database types. This provides the database and schema independence. Generally, converting an object-oriented domain schema into relational schema is a time consuming process. ORM software provides this mapping in less time and requires less or almost no coding.

Use the figure shown on slide 19 to explain the support of ORM technology, such as JPA support for Java objects to database tables.

Tell them that entities are annotated Plain Old Java Objects (POJOs). They are mapped to the tables in a relational database. While working with entities, it is required to declare an entity explicitly. This is done to distinguish it from other regular Java objects that might be used in the application. Java annotations or XML is used to define the mapping of entity to existing database tables.

The JPA specification provides the persistence manager which manages the life cycle of an entity in the persistence context. It also provides the interfaces to execute both static and dynamic queries.

Slide 20

Let us understand JPA specification.

JPA Specification

- ❖ Features of JPA specification are as follows:
 - ❑ Supports POJO programming model for persisting objects into the database.
 - ❑ Provides a standard ORM API which incorporates concepts present in third-party persistence frameworks.
 - ❑ For example, Hibernate, Toplink, Java Data Object (JDO), and so on.
 - ❑ Defines a service provider interface implemented by different persistence providers.

POJOs persisted in the database by JPA are also referred as entities.

© Aptech Ltd. Web Component Development Using Java/Session 5 20

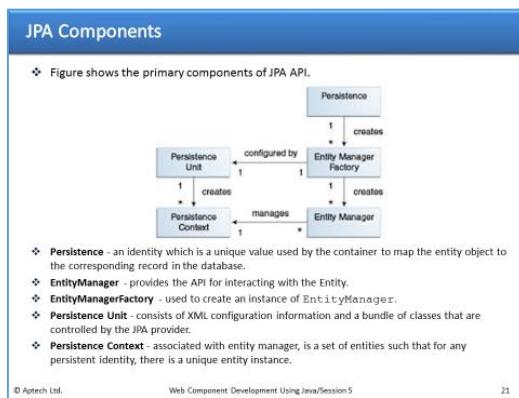
Use slide 20 to explain the features of JPA specification.

Explain them the features of JPA specification:

- It supports POJO programming model for persisting objects into the database. The POJOs persisted in the database by JPA are also referred as entities.
- It provides a standard ORM API which incorporates concepts present in third-party persistence frameworks such as Hibernate, Toplink, Java Data Object (JDO), and so on.
- It defines a service provider interface implemented by different persistence providers. Thus, any change in persistence provider does not affect the entities developed in the application.

Slide 21

Let us understand JPA components.



Use slide 21 to explain the components of JPA specification.

Use the figure shown on slide 21 to display and explain the primary components of JPA API.

Tell them that the primary components of JPA are:

- PersistenceUnit
- EntityManager
- EntityManagerFactory
- PersistenceContext

Tell the students that the role of EntityManager API is to manage entity life cycle instances. It is the core component of the JPA API and is used to create, retrieve, update, and delete data from a database.

The EntityManagerFactory is used to create an instance of EntityManager in your application.

An EntityManager instance is linked with a persistence context. A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance. The entity instances and their life cycle is managed within the persistence context. In other words, persistence context is the connection between the in-memory instances and the database. The EntityManager API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities.

Then, tell them that the set of entities that can be managed by a given EntityManager instance is defined by a persistence unit. A persistence unit defines the set of all classes that are related or grouped by the application and which must be collocated in their mapping to a single database. This persistence unit is defined in a special descriptor file named `persistence.xml`. This file contains logical grouping of entity classes, their metadata mappings, and configurations related to a database.

Slide 22

Let us understand the requirements of entities class in JPA.

Requirements of Entities Class in JPA

- ❖ The entity class must be annotated with `@Entity` annotation defined in `javax.persistence` package.
- ❖ The class must have a no-argument constructor with public or protected access modifier.
- ❖ The properties or methods of the entity class should not be declared as final. The class also should not be declared as final.
- ❖ If the entity instance is passed as parameter to a Session Bean remote interface, then it must implement the `Serializable` interface.
- ❖ Entities can be inherited from entity or non-entity classes.
- ❖ The instance variables of entity class must be declared as private or protected and are accessed only by the methods present in the class.
- ❖ Each entity instance must be identified by a unique identifier or a primary key. The primary key helps the client to access the entity instance.
- ❖ The `@Id` annotation applied on the entity property specifies that the property is a primary key.

© Aptech Ltd. Web Component Development Using Java/Session 5 22

Use slide 22 to explain the requirements of entities class in JPA.

Tell the students that in JPA, persistent domain objects are referred as Entities. Entities are plain old Java objects that are persisted to relational databases or legacy systems. Typically, an entity represents a table in the database and each instance of an entity maps to a row in that table. The entities store data as fields or properties and have getter/setter methods associated with each of the fields present in the table. For example, the employee details can be stored in an entity named **Employee** entity. An instance of `EntityManager` is required to create the entities in the database.

Then, explain the requirements of entity class mentioned on slide 22.

Slides 23 and 24

Let us understand managing entities in the Persistence Context.

Managing Entities in the Persistence Context 1-2

- ❖ To manage the entity instances in the persistence context, an interface named EntityManager API is used.
- ❖ The EntityManager interface is defined in javax.persistence package and is used to interact with the persistence context.
- ❖ The EntityManager interface provides methods to synchronize the state of the entity instance to the database.
- ❖ Figure shows the working of EntityManager in JPA.

```

graph LR
    EMF[EntityManagerFactory] -- "Opens the database" --> EM[EntityManager]
    EM --> ET[EntityTransaction]
    EM --> Q[Query]
    ET -- "manages database transactions" --> ET
  
```

© Aptech Ltd. Web Component Development Using Java/Session 5 23

Managing Entities in the Persistence Context 2-2

- ❖ Table lists some of the methods defined in the EntityManager interface.

Methods	Description
public void persist (Object entity)	The method saves an entity into the database and makes the entity managed.
public <T> T merge (T entity)	The method merges an entity to the EntityManager's persistent context and returns the merged entity.
public void remove (Object entity)	The method removes an entity from the database.
public void flush()	The method synchronizes the state of an entity in the EntityManager's persistent context with the database.

© Aptech Ltd. Web Component Development Using Java/Session 5 24

Use slides 23 and 24 to explain how to manage entities in the persistence context.

Tell the students that a persistence context represents a set of managed entity instances that exist in a particular data store. It is a connection between the entity instances present in the memory and the database. When the entity instance is newly created, it does not have any persistence context associated with it.

At this stage, even the database does not have information about them. When entities are persisted or retrieved from the database, they are associated with the persistence context.

Use the figure shown on slide 23 to display and explain the working of EntityManager in JPA. Use the table as shown on slide 24 to list and explain some of the methods defined in the EntityManager interface.

In-Class Question:

After you finish explaining persistence units, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What is the name of the special descriptor file in which the persistence unit is defined?

Answer:

`persistence.xml`

Slides 25 and 26

Let us understand packaging and deploying entities classes.

Packaging and Deploying Entity Classes 1-2

❖ Persistence Unit:

- Is a set of class mapped to a particular database.
- Is defined in a special descriptor file named `persistence.xml`.
- Contains logical grouping of entity classes, their metadata mappings, and configurations related to a database.
- The configuration details in the `persistence.xml` are used while obtaining the `EntityManager` instance in the client program.
- The `EntityManager` instance accordingly persists the entity instances in the database.

- ❖ The code snippet displays the code for a `persistence.xml` file developed as part of the enterprise application.

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence
    xmlns="http://java.sun.com/xml/ns/persistence"
    <persistence-unit name="intro" />
/>
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

25

Packaging and Deploying Entity Classes 2-2

- ❖ Some of the attributes defined in the `<persistence-unit>` element are as follows:
 - `<description>` - describes the persistence unit and is optional.
 - `<provider>` - must be present in the J2SE environment or when the application requires a provider specific behavior.
 - `<transaction-type>` - has a value either as Java Transaction API (JTA) or RESOURCE_LOCAL or by default, the value is JTA.
 - `<jta-data-source>/<non-jta-data-source>` - used for specifying the Java Naming and Directory Interface (JNDI) name of the data source. JNDI is used by the persistence provider.
 - `<mapping-file>` - contains a list of one or more additional XML files that are used for O/R mapping. The mapping file is used to list the entity classes that are available in the persistence unit.
 - `<properties>` - specifies the configuration properties that are vendor-specific for the persistence unit. Any properties that are not recognized by the persistence provider are ignored.

© Aptech Ltd.

Web Component Development Using Java/Session 5

26

Use slides 25 and 26 to explain packaging and deploying of entities in the servlet application.

Tell the students that the packaging and deploying of an entity class is done using persistence units. A persistence unit is a set of class mapped to a particular database. It is defined in a special descriptor file named `persistence.xml` for a Web application. The configuration details in the `persistence.xml` are used while obtaining the `EntityManager` instance in the client program.

Use the code snippet as shown on slide 25 to explain the code for a `persistence.xml` file developed as part of the enterprise application. Tell them that, in the code, the `persistence.xml` file can contain one or more `<persistence-unit>` elements under the root element, `<persistence>`. Each `<persistence-unit>` element defines the configuration for a single data source. Each `<persistence-unit>` element is identified by the `name` attribute. The `name` attribute is mapped to the `unitName` attribute of the `@PersistenceContext` annotation used in the client program.

Use slide 26 to explain some of the attributes defined in the `<persistence-unit>` element. These are as follows:

- `<description>` - It describes the persistence unit and is optional.
- `<provider>` - It must be present in the J2SE environment or when the application requires a provider specific behavior.
- `<transaction-type>` - It has a value either as Java Transaction API (JTA) or `RESOURCE_LOCAL` or by default, the value is JTA.
- `<jta-data-source>/<non-jta-data-source>` - It is used for specifying the Java Naming and Directory Interface (JNDI) name of the data source. JNDI is used by the persistence provider.
- `<mapping-file>` - It contains a list of one or more additional XML files that are used for O/R mapping. The mapping file is used to list the entity classes that are available in the persistence unit.
- `<properties>` - It specifies the configuration properties that are vendor-specific for the persistence unit. Any properties that are not recognized by the persistence provider are ignored.

Slides 27 to 34

Let us understand accessing database using JPA.

Accessing Database Using JPA 1-8

- The code snippet shows a JSP file, addPlayers.jsp, with a form to add new records to the list.

```
<body>
<h1>New Player record</h1>
<form id="addNewPlayersForm" action="addPlayers"
method="post">

<tr><td>Rank</td><td><input type="text" id = "rank"
name="rank" /></td></tr>

<tr><td>Player</td><td><input type="text" id =
"playerName" name="playerName" /></td></tr>

<tr><td>Team</td><td><input type="text" id = "team"
name="team" /></td></tr>
</table>

<input type="submit" id="CreateRecord"
value="CreateRecord" />
</form>
<a href="DisplayPlayers"><b>Complete List</b></a>
</body>
</html>
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

27

Accessing Database Using JPA 2-8

- The code snippet shows the code to display the records, DisplayPlayers.jsp, which are also accessible from other pages.

```
<body>
<h1>ICC <b>ODI</b> Players Ranking List</h1>
<table id="PlayerTBLList" border="2" bgcolor="#B0B0B0">
<tr><th>Rank</th>
<th>Player</th>
<th>Team</th>
</tr>
<c:forEach var="cricketer" begin="0"
items="${requestScope.playerList}">
<tr>
<td>${cricketer.rank}&ampnbsp</td>
<td>${cricketer.playerName}&ampnbsp</td>
<td>${cricketer.team}&ampnbsp</td>
</tr>
</c:forEach>
</table>
<a href="addPlayers.jsp"><b>Add New Player Record</b></a>
</body>
</html>
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

28

Accessing Database Using JPA 3-8

- The code snippet shows the index.jsp page.

```
<html>
<head>
</head>
<body>
<jsp:forward page="DisplayPlayers" />
</body>
</html>
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

29

Accessing Database Using JPA 4-8

- The code snippet shows the entity class named Players.

```
@Entity
@Table(name = "PLAYERS")
public class Players {
    // column mapping
    @Rank
    @Column(name = "RANK")
    private String rank;
    @Column(name = "PLAYER")
    private String playerName;
    @Column(name = "TEAM")
    private String team;

    public Players() { }
    public Players(String rank, String playerName, String team)
    {
        this.rank = rank; this.playerName = playerName;
        this.team = team;
    }

    public String getRank() {
        return this.id;
    }
    public String getPlayer() {
        return this.playerName;
    }
    public String getTeam () {
        return this.team;
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

30

Accessing Database Using JPA 5-8

- The code snippet shows the Servlet, addNewPlayersServlet.java, for adding new players.

```
...
protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException {
    assert eManagerFactory != null;
    EntityManager eManager = null;
    try {
        // the data from user's form
        String rank = (String) request.getParameter("rank");
        String playerName = (String) request.getParameter("playerName");
        String team = (String) request.getParameter("team");
    }
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

31

Accessing Database Using JPA 6-8

```
// player instance
Players player = new Players(rank, playerName, team);
// begin transaction uTransaction
uTransaction.begin();

// since the eManager is created inside a transaction, it is
// associated with the transaction
eManager = eManagerFactory.createEntityManager();

//persist the player entity
eManager.persist(player);

// commit transaction uTransaction
uTransaction.commit();

request.getRequestDispatcher("DisplayPlayers").forward(request,
response);
} catch (Exception ex) {

}
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

32

Accessing Database Using JPA 7-8

- The code snippet shows the code for displaying players data in the Servlet, `DisplayPlayersServlet.java`.

```

        EntityManager eManager = null;
        try {
            eManager = eManagerFactory.createEntityManager();
            //query for all the players in database
            List players = eManager.createQuery("select p from Player p")
                .getResultsList();
            request.setAttribute("playerList", players);
            //Forward to the jsp page for rendering
            request.getRequestDispatcher("displayPlayers.jsp").forward(request, response);
        } catch (Exception ex) {
            throw new ServletException(ex);
        } finally {
            //close the eManager to release any resources held up by
            //the persistent provider
            if (eManager != null) {
                eManager.close();
            }
        }
    }
}

```

© Aptech Ltd. Web Component Development Using Java/Session 5 33

Accessing Database Using JPA 8-8

- Figures display the JSP page with the index page and other pages in the execution of the players application.

© Aptech Ltd. Web Component Development Using Java/Session 5 34

Use slides 27 to 34 to explain accessing database using JPA with an example.

Tell the students to consider an example for inserting and displaying of records of the players from the database. The Web application containing JSP pages, entities, and Servlet classes is created for performing operations on the players details stored in the database.

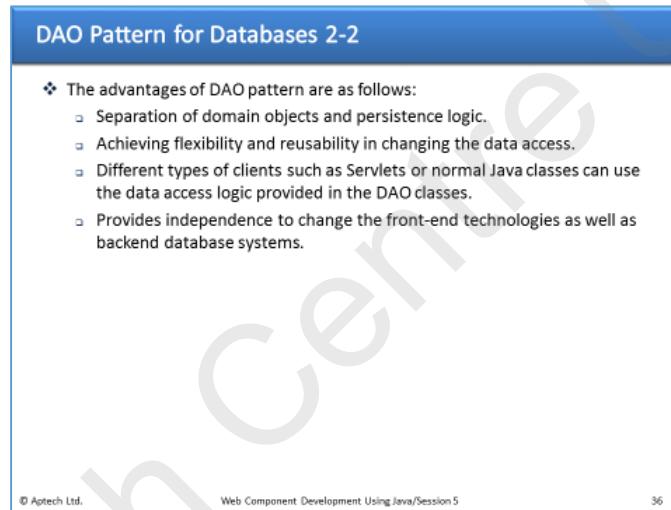
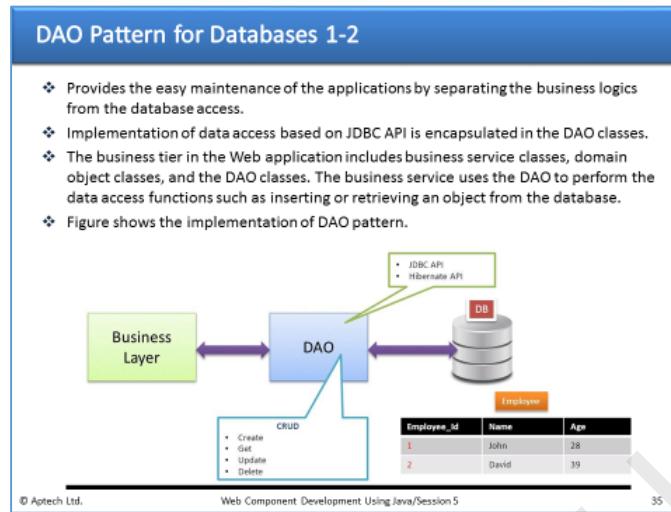
Use the code snippet as shown on slides 27 to 29 to explain the creation of JSP pages such as adding records to the players list, displaying the records of the players on the Web page, and `index.jsp` page.

Use the code snippet as shown on slide 30 to show the entity class named **Players**. Tell them that the code snippet defines a Java Persistence API entity class.

Use the code snippet as shown on slides 31 and 32 to explain the servlet program for adding new player details to the database. Tell them that `EntityManagerFactory` opens the database when it is constructed. An `EntityManagerFactory` supports the instantiation of `EntityManager` instances. The `EntityManager` instance is used to persist the player entity in the database. Note that while persisting the entity, a transaction boundary is implemented by using `UserTransaction` interface. Then, the request is dispatched to the `DisplayPlayers` servlet for displaying the player details. Then, use the code snippet as shown on slide 33 to explain displaying of players details in the servlet. Use the figure shown on slide 34 to show the execution of the application.

Slides 35 and 36

Let us understand DAO Pattern for databases.



Use slides 35 and 36 to explain DAO pattern for databases.

Tell the students that Data Access Object (DAO) pattern is used for separating low-level data accessing API or operations from high-level business services. Thus, providing the independence and increasing flexibility between the layers in the application development.

Then, explain them that the participants in DAO pattern are:

- DAO Interface – It defines the processes to be executed on a model object (s).
- DAO concrete class – It is responsible to get data from a data source. The data source can be a database, XML, or any other storage mechanism.
- Value Object or Model Object - A POJO comprising getter/setter methods to store data recovered using DAO class.

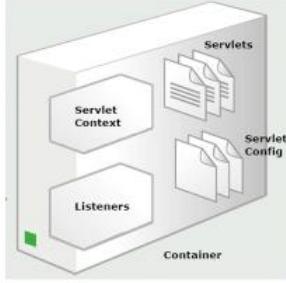
Then, using the figure shown on slide 35, explain the implementation of DAO pattern. Then, use slide 36 to describe the advantages of DAO pattern.

Slide 37

Let us understand session event handling.

Session Event Handling

- ❖ Session handling is used to maintain the state of a user till the lifetime of the session.
- ❖ In Java EE,
 - The Web container transfers control to event handlers i.e. event listeners to listen events related to Servlets, JSP, and so on.



The diagram illustrates the components of a Java EE web application within a container. It shows a central 'Container' box containing four main components: 'Servlets' (represented by a folder icon), 'Servlet Context' (represented by a document icon), 'Listeners' (represented by a person icon), and 'Servlet Config' (represented by a folder icon).

© Aptech Ltd. Web Component Development Using Java/Session 5 37

Use slide 37 to explain session event handling.

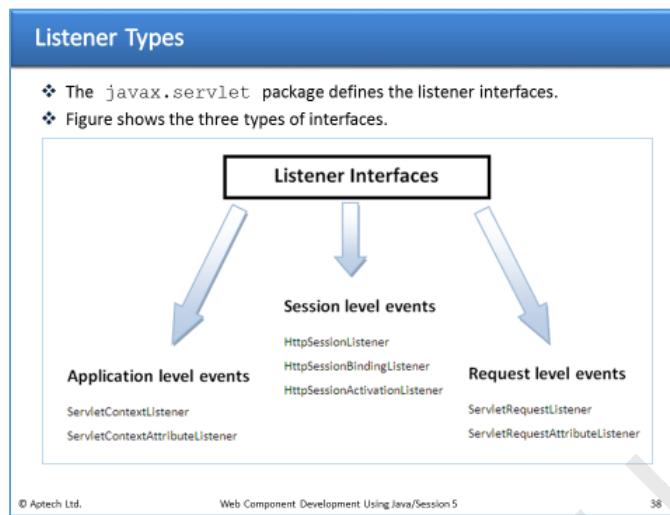
In many situations, it is desirable to be notified about container-managed life cycle events. Some of the examples can be when the container initializes a Web application or when the container destroys the Web application. The information about the events occurring in the container can be helpful in deciding how other resources dependent on them will perform the task.

Handling of events in the operating systems are programmed. Interrupt handlers are used as event handlers that are used for handling events generated by the CPU. Event driven programs are made up of functions, which listen and handle events. For example, act of clicking a mouse. In Java EE, the Web container transfer control to event handlers that are responsible for receiving and handling of events related to Servlets, JSP, and so on.

Use the figure shown on slide 37 to show servlets and listeners designed as a part of Web application deployed in the container.

Slide 38

Let us understand listener types.

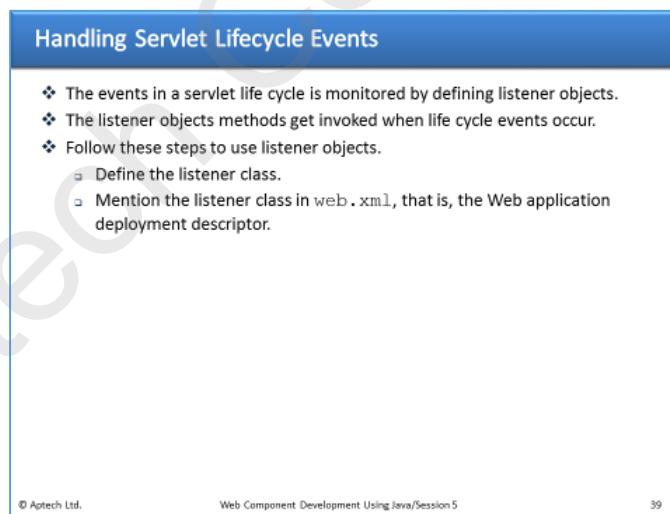


Use slide 38 to explain listener types.

Use the figure shown on slide 38 to explain the three types of listener interfaces. Tell them that listeners can be categorized as application level events, session level events, and request level events.

Slide 39

Let us understand handling servlet life cycle events.



Use slide 39 to explain handling servlet life cycle events.

Tell the students that in Web development, listeners assist by recognizing application events. These are also known as observers or event handlers. This helps developers in controlling life cycle of `ServletContext`, `ServletRequest`, and `HttpSession` objects. The events in a servlet life cycle is monitored by defining listener objects. These object methods get invoked when life cycle events occur in the container.

Tell them that to use listener objects, they have to follow these steps:

- Define the listener class.
- Mention the listener class in `web.xml`, that is, the Web application deployment descriptor.

The servlet context provides access to various resources and facilities, which are common to all Servlets in the application. The Servlet API is used to set the information common to all servlets in an application. Servlets running in the same server sometimes share resources, such as JSP pages, files, and other servlets. This is required when several servlets are bound together to form a single application, such as a chat application, which creates single chat room for all users. The attributes of `ServletContext` class can be used to share information among a group of Servlets.

Slide 40

Let us understand ServletContextListener.

ServletContextListener

- ❖ The methods present in the `ServletContextListener` class are as follows:
 - ❑ `contextInitialized()` - returns the notification that a Web application initialization has started.
 - ❑ `contextDestroyed()` - returns the notification about closing of the servletcontext. All servlets and filters are closed before notification.
- ❖ The code snippet shows the use of `ServletContext` methods.


```
public void contextInitialized(ServletContextEvent event) {
    ServletContext context = event.getServletContext();
    String IP = "10.1.1.142";
    context.setAttribute("DefaultAddress", IP);
}

public void contextDestroyed(ServletContextEvent event) {
    ServletContext context = event.getServletContext();
    String IP = context.getAttribute("DefaultAddress");
}
```

© Aptech Ltd. Web Component Development Using Java/Session 5 40

Use slide 40 to explain ServletContextListener.

Tell the students that the interface `ServletContextListener` extends `java.util.EventListener` interface. This is the interface for receiving notification events about `ServletContext` life cycle changes. To receive these notification events, the implementation class must be either declared in the deployment descriptor of the Web application or registered by means of any of the `addListener` methods defined on `ServletContext`.

Explain them with the syntax of the methods present in `ServletContextListener` class as:

- For `contextInitialized()` method - `public void contextInitialized(ServletContextEvent sce)`
- For `contextDestroyed()` method - `public void contextDestroyed(ServletContextEvent sce)`

Use the code snippet as shown on slide 40 to explain the use of `ServletContext` methods.

Slides 41 and 42

Let us understand ServletContextEvent.

ServletContextEvent 1-2

- ❖ The method present in ServletContextEvent class is as follows:
 - `getServletContext()` - returns the modified servlet context.
- ❖ The code snippet shows the use of the method `getServletContext()`.


```
// Save and get an application-scoped value
getServletContext().setAttribute("app-param", "app-
value1");

value1 = getServletContext().getAttribute("app-param");
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

41

ServletContextEvent 2-2

- ❖ Figures depict the hierarchy for ServletContextListener.

```

classDiagram
    class java.util.EventListener
    class ServletContextListener {
        <<java.util.EventListener>>
        <<ServletContextListener>>
        <<contextInitialized(), contextDestroyed()>>
    }
    class java.util.EventObject {
        <<java.util.EventObject>>
        <<ServletContextEvent>>
        <<getServletContext()>>
    }
    java.util.EventListener --> ServletContextListener
    ServletContextListener --> contextInitialized()
    ServletContextListener --> contextDestroyed()
    java.util.EventObject --> ServletContextEvent
    ServletContextEvent --> getServletContext()
  
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

42

Use slides 41 and 42 to explain ServletContextEvent. Tell the students that the `ServletContextEvent` class extends the `java.util.EventObject` class. This event class notifies about changes made to a Servlet context in a Web application.

Explain them with the syntax of `getServletContext()` as: `public ServletContext getServletContext()`.

Use the code snippet given on slide 41 to display the use of `getServletContext()` method.
Use the figure in slide 42 to depict the hierarchy for `ServletContextListener`.

Slides 43 to 45

Let us understand ServletContextAttributeListener.

ServletContextAttributeListener 1-3

- ❖ The methods present in the interface are as follows:
 - ❑ **attributeAdded()**
 - It notifies that a new attribute is added to the ServletContext.
 - It is invoked after an attribute has been added.
 - ❑ **attributeRemoved()**
 - It notifies that an attribute is removed from ServletContext.
 - It is invoked after an attribute has been removed.

© Aptech Ltd. Web Component Development Using Java/Session 5 43

ServletContextAttributeListener 2-3

- ❖ The code snippet shows the method of the ServletContextAttributeListener.


```
public void attributeAdded(ServletContextAttributeEvent event) {
    log.append(event.getName() + "," + event.getValue()
    + "," + new Date() + "\n");
}

public void attributeRemoved
(ServletContextAttributeEvent event){
    log.append(event.getName() + "," + event.getValue()
    + value1 = " , " + new Date() + "\n");
}
```
- ❖ **attributeReplaced()** - notifies that an attribute of ServletContext has been replaced. The method is invoked after an attribute has been replaced.

© Aptech Ltd. Web Component Development Using Java/Session 5 44

ServletContextAttributeListener 3-3

- ❖ Figure depicts the methods of ServletContextAttributeListener.

```

classDiagram
    class java.util.EventListener
    class ServletContextAttributeListener {
        <<attributeAdded()>>
        <<attributeRemoved()>>
        <<attributeReplaced()>>
    }
    EventListener --> ServletContextAttributeListener
  
```

© Aptech Ltd. Web Component Development Using Java/Session 5 45

Use slides 43 to 45 to explain the methods present in the ServletContextAttributeListener Interface.

Tell the students that the interface extends `java.util.EventListener` interface. The interface receives a notification about any modifications made to the attribute list on the servlet context of a Web application. The implementation class must be configured in the deployment descriptor for the Web application to receive notification events.

Explain them with the syntax of the methods present in `ServletContextAttributeListener` interface as:

- `public void attributeAdded(ServletContextAttributeEvent sca1)`
- `public void attributeRemoved(ServletContextAttributeEvent sca1)`
- `public void attributeReplaced(ServletContextAttributeEvent`

Then, explain the code snippet as shown on slide 44 to work with the method of the `ServletContextAttributeListener`.

Use the figure shown on slide 45 to depict the methods of `ServletContextAttributeListener`.

Slides 46 and 47

Let us understand ServletContextAttributeEvent.

ServletContextAttributeEvent 1-2

- ❖ The methods present in the class are as follows:
 - `getName()` - name of the altered attribute in the `ServletContext` is returned.
 - The code snippet gets the name of the attribute and checks whether the attribute is altered.

```
public void attributeRemoved(ServletContextAttributeEvent
scae1) {
    if (scae1.getName().equals("heading")) {
        ServletContext context1 =
            scae1.getServletContext();
        String sHeading1=(String)
            context1.getAttribute("heading");
        context.log("Heading Replaced="+sHeading1);
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 5 46

ServletContextAttributeEvent 2-2

- `getValue()` - attribute value that has been added, removed, or replaced is returned.
- The code snippet gets the value of the attribute and checks whether the value is not equal to null.

```
HttpSession session = request.getSession(true);
ShoppingCart1 previousItems1 =
(ShoppingCart1)session.getValue("previousItems1");
if (previousItems1 != null) {
    doSomethingWith(previousItems1);
} else {
    previousItems1 = new ShoppingCart1(...);
    doSomethingElseWith(previousItems1);
}
```

© Aptech Ltd. Web Component Development Using Java/Session 5 47

Use slides 46 and 47 to explain the methods present in the `ServletContextAttributeEvent` class.

Tell the students that the class extends `ServletContextEvent` class, and this is the event class for notifications about changes to the attributes of the `ServletContext` of a Web application.

Explain them with the syntax of the methods present in `ServletContextAttributeEvent` class as:

- `public java.lang.String getName()`
- `public java.lang.Object getValue()`

Use the code snippet as shown on slide 46 to display the code that gets the name of the attribute and checks whether the attribute is altered.

Use the code snippet as shown on slide 47 to display the code that gets the value of the attribute and checks whether the value is not equal to null.

Tips:

The following table shows the servlet context events, the interfaces that must be implemented in the event listener class, and the event methods invoked when the servlet context method occurs.

Event	Interface	Event Method
Servlet context is created.	javax.servlet.ServletContextListener	contextInitialized()
Servlet context is about to be shut down.	javax.servlet.ServletContextListener	contextDestroyed()
An attribute is added.	javax.servlet.ServletContextAttributeListener	attributeAdded()
An attribute is removed.	javax.servlet.ServletContextAttributeListener	attributeRemoved()
An attribute is replaced.	javax.servlet.ServletContextAttributeListener	attributeReplaced()

Slides 48 and 49

Let us understand HttpSessionAttributeListener.

HttpSessionAttributeListener 1-2

The methods belonging to this interface are as follows:

- ❖ **attributeAdded()** – notifies whenever there is an addition of new attributes to a session, and is called after the addition of attribute.
- ❖ **attributeRemoved()** – notifies whenever there is a removal of attributes from a session, and is called after the removal of attribute.

© Aptech Ltd. Web Component Development Using Java/Session 5 48

HttpSessionAttributeListener 2-2

- ❖ **attributeReplaced()** - notifies whenever attributes are replaced in a session, and is called after the attribute is replaced.
- ❖ The code snippet implements HttpSessionAttributeListener interface and its methods.

```
public class UserAttributeListener implements
HttpSessionAttributeListener{
// Declaring attributeAdded() method
public void attributeAdded(HttpSessionBindingEvent Demoevent)
{
// Logging the event details
HTTPSession session = Demoevent.getSession();
ServletContext sc = session.getServletContext();
sc.log(Demoevent.getName() + "," +
Demoevent.getValue());
}
// Declaring attributeRemoved() method
public void attributeRemoved(HttpSessionBindingEvent Demoevent) {}
// Declaring attributeReplaced() method
public void attributeReplaced(HttpSessionBindingEvent Demoevent) {}
}
```

© Aptech Ltd. Web Component Development Using Java/Session 5 49

Use slides 48 and 49 to explain the methods belonging to HttpSessionAttributeListener interface. Tell the students that this interface extends the `java.util.EventListener` class. It is called whenever some changes are made to the attribute list on the servlet session of a Web application. HttpSessionAttributeListener is used to notify these changes. The listener is used to notify when an attribute has been added, removed, or replaced by another attribute.

Explain them with the syntaxes of the methods belonging to HttpSessionAttributeListener interface as:

- `public void attributeRemoved(HttpSessionBindingEvent se)`
- `public void attributeRemoved(ServletContextAttributeEventsca1)`
- `public void attributeReplaced(HttpSessionBindingEvent se)`

Use the code snippet as shown on slide 49 to explain the code that implements `HttpSessionAttributeListener` interface and its methods.

Slide 50

Let us understand `HttpSessionBindingListener`.

HttpSessionBindingListener

The methods belonging to this interface are as follows:

- ❖ `valueBound()` - notifies the object on being bound to a session and is responsible for identification of the session.
- ❖ `valueUnbound()` - notifies the object on being unbound from a session and is responsible for identification of the session.
- ❖ The code snippet implements the `valueBound` and `valueUnbound` methods of `HttpSessionBindingListener` interface.

```
public class UserInfo implements HttpSessionBindingListener{
    private String uName;
    public User(String userName){
        uName = userName;
    }
    public void valueBound(HttpSessionBindingEvent ev){
        System.out.println(uName+" user bound");
    }
    public void valueUnbound(HttpSessionBindingEvent ev){
        System.out.println(uName+" user unbound");
    }
}
```

© Aptech Ltd.
Web Component Development Using Java/Session 5
50

Use slide 50 to explain the methods belonging to `HttpSessionBindingListener` interface.

Tell the students that the `HttpSessionBindingListener` interface notifies the object when it is being bound to or unbound from a session. This notification can be the result of a forced unbinding of an attribute from a session by the programmer, invalidation of the session, or due to timing out of the session. The implementation class do not require any configuration within the deployment descriptor of the Web application.

Explain them with the syntaxes of the methods belonging to `HttpSessionBindingListener` interface as:

- `public void valueBound(HttpSessionBindingEvent event)`
- `public void valueUnbound(HttpSessionBindingEvent event)`

Use the code snippet as shown on slide 50 to display the code that implements the `valueBound` and `valueUnbound` methods of `HttpSessionBindingListener` interface.

Slides 51 to 53

Let us understand HttpSessionBindingEvent.

HttpSessionBindingEvent 1-3

The methods belonging to this class are as follows:

- ❖ **getName ()** - returns a string specifying the name with which the attribute is bound to or unbound from the session.
- ❖ The code snippet gets the name string specifying the name with which the attribute is bound to or unbound from the session.

```
private void checkAttribute(HttpSessionBindingEvent event,
String orderAttributeName, String keyItemName,
String message) {

String demoCurrentAttributeName = event.getName();
String demoCurrentItemName = (String)event.getValue();
if (demoCurrentAttributeName.equals(orderAttributeName) &&
demoCurrentItemName.equals(keyItemName)) {
    ServletContext context =
event.getSession().getServletContext();
context.log("Customer" + message + keyItemName + ".");
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

51

HttpSessionBindingEvent 2-3

- ❖ **getSession ()** - returns the session object that has changed.
- ❖ The code snippet returns the session object that has changed.

```
public void sessionCreated(HttpSessionBindingEvent event)
{
    // Retrieves modified session object
    HttpSession session = event.getSession();
    session.getServletContext().log("SessionListener01:
sessionCreated(" + session.getId() + ")");
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

52

HttpSessionBindingEvent 3-3

- ❖ **getValue ()** - returns the value of the attribute that has been added, removed, or replaced.
- ❖ The code snippet returns the value of the attribute that has been added, removed, or replaced.

```
private void checkAttribute(HttpSessionBindingEvent event,
String orderAttributeName, String keyItemName,
String message) {

String demoCurrentAttributeName = event.getName();
String demoCurrentItemName = (String)event.getValue();

if (demoCurrentAttributeName.equals(orderAttributeName) &&
demoCurrentItemName.equals(keyItemName)) {
    ServletContext context =
event.getSession().getServletContext();
context.log("Customer" + message + keyItemName + ".");
}}
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

53

Use slides 51 to 53 to explain the methods belonging to HttpSessionBindingEvent class.

Tell the students that the `HttpSessionBindingEvent` event is sent on occasions to an object that implements `HttpSessionBindingListener` when being bound to or unbound from the session.

It is also sent when `HttpSessionAttributeListener` is configured during the deployment descriptor while being bound, unbound, or replaced in a session. The object is bound to a session by a call to `HttpSession.setAttribute` and unbounded from a session by a call to `HttpSession.removeAttribute`.

Explain them with the syntax of the methods belonging to `HttpSessionBindingEvent` as:

- `public java.lang.String getName()`
- `public HttpSession getSession()`
- `public java.lang.Object getValue()`

Use the code snippet as shown on slide 52 to display the code that returns the session object that has changed.

Use the code snippet as shown on slide 53 to display the code that returns the value of the attribute that has been added, removed, or replaced.

Slides 54 and 55

Let us understand HttpSessionListener.

HttpSessionListener 1-2

The methods belonging to this interface are as follows:

- ❖ **sessionCreated(HttpSessionEvent se)** – provides notification that a session was created.
- ❖ The code snippet implements HttpSessionListener interface and its methods.

```
public class SessionTracker extends HttpServlet
    implements HttpSessionListener{
    static private int sessionCount; // count of session
    // Implementing sessionCreated()
    public void sessionCreated(HttpSessionEvent event) {
        sessionCount++;
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

54

HttpSessionListener 2-2

- ❖ **sessionDestroyed(HttpSessionEvent se)** – provides notification that a session is about to be invalidated.
- ❖ The code snippet implements the method **SessionDestroyed()** of the HttpSessionListener interface.
- ❖ The SessionTracker class must be configured in the deployment descriptor of the Web application as shown in the code snippet.

```
public void sessionDestroyed(HttpSessionEvent se) {
    HttpSession session = se.getSession();
    try{
        session.invalidate();
    } catch(IllegalStateException e) { }
}
```

```
<listener>
<listener-class>
SessionTracker
</listener-class>
</listener>
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

55

Use slides 54 and 55 to explain the methods belonging to HttpSessionListener interface.

Tell the students that the changes to the list of active sessions in Web application are notified to the implementations of HttpSessionListener. The deployment descriptor for the Web application must have the configured implementation class to receive the notification events. Various session events are session creation and destruction.

Explain them with the syntaxes of the methods belonging to HttpSessionListener interface as:

- `public void sessionCreated(HttpSessionEvent se)`
- `public void sessionDestroyed(HttpSessionEvent se)`

Use the code snippet as shown on slide 54 to show the code that implements HttpSessionListener interface and its methods.

Then, explain the code snippet as shown on slide 55 to explain the code that implements the method `SessionDestroyed()` of the `HttpSessionListener` interface.

Then, explain the next code snippet as shown on slide 55 to tell them that the `SessionTracker` class must be configured in the deployment descriptor of the Web application.

Slide 56

Let us understand HttpSessionActivationListener.

HttpSessionActivationListener

The methods present in the listener are as follows:

- ❖ **sessionDidActivate (HttpSessionEvent se)** – provides notification that the session has just been activated.
- ❖ **sessionWillPassivate (HttpSessionEvent se)** – provides notification that the session is about to be passivated.
- ❖ The code snippet implements HttpSessionActivationListener interface and its methods.

```
public class SessionTracker implements HttpSessionActivationListener {
    public void sessionWillPassivate(HttpSessionEvent se) {
        System.out.println("session is about to be passivated");
    }

    public void sessionDidActivate(HttpSessionEvent se) {
        System.out.println("session has just been activated");
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 5 56

Use slide 56 to explain the methods present in HttpSessionActivationListener.

Tell the students that sometimes to manage the size of its working set, idle stateful session bean instance is transferred temporarily to the secondary storage. The transfer from the working set to secondary storage is called instance passivation. The transfer back is called activation. The objects bounded to a session might listen to container events at the same time notifying them about the probability of session being passivated or activated.

Tell them that the `HttpSessionActivationListener` is implemented when a container migrates the sessions between virtual machines or persists sessions, if required, to provide notification to all attributes bounded sessions. The implementation class do not require any configuration within the deployment descriptor of the Web application.

Explain them with the syntax of the methods present in `HttpSessionActivationListener` as:

- `public void sessionCreated(HttpSessionEvent se)`
- `public void sessionWillPassivate(HttpSessionEvent se)`

Use the code snippet as shown on slide 56 to explain the code that implements `HttpSessionActivationListener` interface and its methods.

Slides 57 and 58

Let us understand HttpSessionEvent.

HttpSessionEvent

The method belonging to this class is getSession().
 ✤ **getSession** – returns the session that changed within a Web application.

❖ The code snippet shows the getSession() method.

```
// returns modified session object//
public HttpSession getSession()
{
    return session;
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

57

Use of Listeners for Handling Events

❖ Event listeners should be executed quickly, because all event listening and drawing methods are executed in the same thread.
 ❖ While designing the program, one should implement their Event Listeners in a class that is not public.
 ❖ The code snippet shows the implementation of actionPerformed() method.

```
public class Beeper ... implements ActionListener {
    ...
    //here initialization:
    button.addActionListener(this);
    ...
    public void actionPerformed(ActionEvent e) {
        ...
        //Make a beep sound...
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 5

58

Use slides 57 and 58 to explain HttpSessionEvent class.

Tell the students that the notifications for the changes to the sessions within a Web application are represented by HttpSessionEvent .

Explain them with the syntax of getSession() method as:
`public HttpSession getSession()`

Use the code snippet as shown on slide 57 to display the getSession() method.

Then, tell them that while designing the program, one should implement their EventListeners in a class that is not public. In case, if more secure implementation is required, then the implementation can be private.

Slide 59

Let us summarize the session.

Summary

- ❖ The most common operation performed by the Servlet is storing and retrieving database information.
- ❖ Java provides various mechanisms using which data can be accessed from the database using JDBC or JPA API.
- ❖ The JDBC API provides classes and interfaces that allow a Web application to access and perform operations on the databases.
- ❖ JPA is an ORM technology that persist the entities in the database.
- ❖ Java annotations or XML are used to define the mapping of entity to existing database tables.
- ❖ In JPA, persistent objects are referred as Entities. Entities are plain old Java objects that are persisted to relational databases or legacy systems.
- ❖ A persistence context represents a set of managed entity instances that exist in a particular data store.
- ❖ The DAO pattern provides the easy maintenance of the applications by separating the business logics from the database access.
- ❖ The events in a servlet life cycle is monitored by defining listener objects. These objects methods get invoked when life cycle events occur.

© Aptech Ltd. Web Component Development Using Java/Session 5 59

In slide 59, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

5.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore asynchronous Servlets and server push mechanism.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 6 – Asynchronous Servlet Communication

6.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

6.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the need of Asynchronous Servlet
- Explain how to create Asynchronous Servlet and Asynchronous Listener
- Explain the concept of server push mechanism
- Explain how to create an asynchronous JavaScript client using XMLHttpRequest object
- Explain the need of non-blocking I/O support in Servlet
- Explain how to implement non-blocking I/O in asynchronous Servlet
- Explain the need for protocol upgrade
- Explain the process of protocol upgrade in a Servlet

6.1.2 Teaching Skills

To teach this session successfully, you should be aware of Asynchronous Servlet. Familiarize yourself with the concept of server push mechanism and also how to create Asynchronous Servlet and Asynchronous Listener.

Aware yourself with non-blocking I/O support in Servlet and how to create an asynchronous JavaScript client using XMLHttpRequest object. You should know how to implement non-blocking I/O in an asynchronous Servlet and the need for protocol upgrade. Aware yourself with the process of protocol upgrade in a Servlet.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❖ Explain the need of Asynchronous Servlet
- ❖ Explain how to create Asynchronous Servlet and Asynchronous Listener
- ❖ Explain the concept of server push mechanism
- ❖ Explain how to create an asynchronous JavaScript client using XMLHttpRequest object
- ❖ Explain the need of non-blocking I/O support in Servlet
- ❖ Explain how to implement non-blocking I/O in asynchronous Servlet
- ❖ Explain the need for protocol upgrade
- ❖ Explain the process of protocol upgrade in a Servlet

© Aptech Ltd. Web Component Development Using Java/Session 6 2

Tell the students that they will be introduced to asynchronous communication and its usage in creating asynchronous servlets. The session explains how to create asynchronous servlet and asynchronous listener.

The session also explains the concept of server push mechanism and also how to create an asynchronous JavaScript client using XMLHttpRequest object. It describes the need of non-blocking I/O support in servlet and how to implement non-blocking I/O in asynchronous servlet.

Finally, the session explains the need for protocol upgrade and the process of protocol upgrade in a servlet.

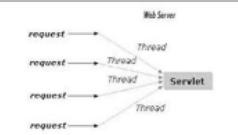
6.2 In-Class Explanations

Slide 3

Let us understand the architecture of the Servlet.

Introduction

- ❖ The architecture of the Servlet is based on **Multithreaded model**.
- ❖ The Web container creates Servlet threads to process the client requests.
- ❖ Each client will receive its dedicated thread which will serve the request and generate a response to be sent back to the client.



- ❖ A thread associated with the client request may not process the request all the time and may sit idle because of the following reasons:
 - Servlet may take a long running task like making a database connection call for query execution or invoking a remote Web service.
 - Servlet may have to wait for some dependent event to proceed for the response generation.

Servlet 3.0 API has provided a new feature in its specification to process the request asynchronously.

© Aptech Ltd. Web Component Development Using Java/Session 6 3

Use slide 3 to explain the architecture of the Servlet.

Consider a situation where the Web server receives multiple client requests for the Servlet. To serve multiple requests, the Web container needs large number of threads in the container pool. A thread associated with the client request may not process the request all the time and may sit idle. This can lead to thread starvation on the server.

Then, discuss some of the situations where the thread can go idle. Some of the reasons, where the thread can go idle while processing the request are as follows:

- Servlet may take a long running task like making a database connection call for query execution or invoking a remote Web service. In these situations, the thread needs to wait for the resource availability, so that it can generate the response to be sent to the client.
- Servlet may have to wait for some dependent event to proceed for the response generation. The situation can occur when waiting to receive information from another client to proceed with the response generation.

Then, explain them that Servlet 3.0 API has provided a new feature in its specification to process the request asynchronously. With Asynchronous processing, one can avoid blocking requests by allowing the thread to perform other operations, while the input is returned to the client.

In-Class Question:

After you finish explaining Servlets and its features, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



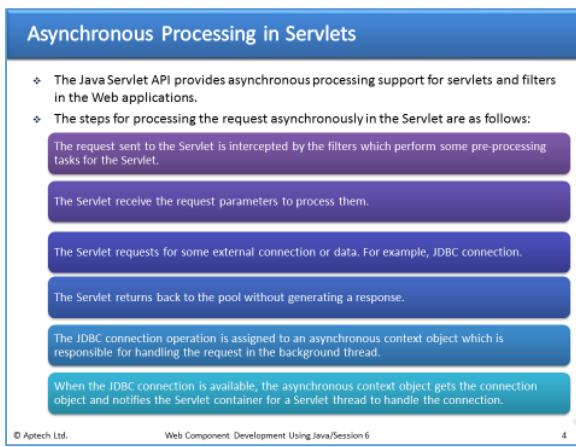
By using asynchronous processing, one can avoid blocking _____.

Answer:

request

Slide 4

Let us understand asynchronous processing in Servlets.



Use slide 4 to explain asynchronous processing in servlets. Tell the students that in asynchronous processing, a servlet thread waiting for a resource is released by the container and returned to the pool.

Asynchronous processing is carried out in the background thread. Once the task is completed, the background thread can generate a result which needs to be dispatched to a Servlet for processing. A notification is sent to the servlet container to handle the generated result.

A different servlet thread available in the pool is allocated to handle the result and interact with the client for sending the generated response.

Then, discuss the steps for processing some long awaiting tasks such as JDBC, such requests can be handled asynchronously as mentioned:

- The request sent to the Servlet is intercepted by the filters which perform some pre-processing tasks for the Servlet. For example, authentication of the user can be pre-processed within a Servlet.
- The Servlet receive the request parameters to process them.
- The Servlet requests for some external connection or data. For example, sends a requests for obtaining a connection with a database. If there are already some requests waiting for the connection, then the Servlet joins a waiting queue for a JDBC connection.
- While waiting for the JDBC connection, the Servlet returns back to the pool without generating a response.
- The JDBC connection operation is assigned to an asynchronous context object which is responsible for handling the request in the background thread.
- When the JDBC connection is available, the asynchronous context object gets the connection object and notifies the Servlet container for a Servlet thread to handle the connection.

Slide 5

Let us understand how to work with asynchronous servlet.

Handling Asynchronous Servlet

- ❖ To support asynchronous processing, the separation of request from the generated response needs to be separated.
- ❖ The `javax.servlet.AsyncContext` class is used to process the request asynchronously within the Servlet.
- ❖ Apart from creating the object of `AsyncContext`, a user need to enable the Servlet class with asynchronous processing.
- ❖ This is done by enabling the attribute `asyncSupported` to true in the `@WebServlet` annotation.

© Aptech Ltd. Web Component Development Using Java/Session 6 5

Use slide 5 to explain handling asynchronous servlet.

To support asynchronous processing, the separation of request from the generated response needs to be separated. This is achieved by using the class `javax.servlet.AsyncContext`. This class is used to process the request asynchronously within the servlet.

Apart from creating the object of `AsyncContext`, you need to enable the Servlet class with asynchronous processing. This is done by enabling the attribute `asyncSupported` to true in the `@WebServlet` annotation. This makes the application server know that the servlet is asynchronous.

In case, if you are using deployment descriptor, `web.xml` file for registering the servlet created in the Web application, then by adding the `asyncSupported` element in the file, we can let the server know that the given servlet is asynchronous.

Slides 6 to 15

Let us understand AsyncContext class.

AsyncContext Class 1-10

- The AsyncContext class provide methods that can be used to obtain the instance of the AsyncContext within the Servlet service() method.
- Table lists the methods of the AsyncContext class.

Method	Description
void start(Runnable run)	Gets a thread from the managed thread pool to run the specified Runnable thread.
ServletRequest getRequest()	Obtains the parameters from the request for the asynchronous context.
ServletResponse getResponse()	Used inside the asynchronous context to write to the response with the results of the blocking operation.
void complete()	Completes the asynchronous operation and closes the response object associated with the asynchronous object.
void dispatch(String path)	Dispatches the request and response objects to the URL to be forwarded to another Servlet thread.

© Aptech Ltd. Web Component Development Using Java/Session 6 6

AsyncContext Class 2-10

- Following are the steps to be performed to handle asynchronous request in the asynchronous Servlet:

Obtain the object of the AsyncContext class by invoking the startAsync() method on the ServletRequest object.

Invoke the asyncContext.start() method by passing a Runnable thread reference which is responsible to execute a running task in the background.

Call setTimeout() on the asyncContext, after the number of milliseconds the container has to wait for the specified task to complete.

Call asyncContext.complete() or asyncContext.dispatch() from the Runnable thread to complete the task or forward the request and response object to some other Servlet.

© Aptech Ltd. Web Component Development Using Java/Session 6 7

AsyncContext Class 3-10

- The code snippet shows the skeleton for obtaining the asynchronous context within the Servlet doGet() or doPost() method.

```

@.WebServlet(urlPatterns={"/asyncservlet"}, asyncSupported=true)

public class AsyncServlet extends HttpServlet {

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException, IOException {
        final AsyncContext = request.startAsync();
        asyncContext.setTimeout(...);

        asyncContext.start(new Runnable() {
            @Override
            public void run() { // long running task
                ...
            }
        });
    }
}

```

© Aptech Ltd. Web Component Development Using Java/Session 6 8

AsyncContext Class 4-10

- The code snippet demonstrates how to process the request asynchronously in the Servlet.

```
@WebServlet(name = "AsyncDispatchServlet", urlPatterns = { "/asyncDispatch" }, asyncSupported = true)
public class AsyncDispatchServlet extends HttpServlet {
    @Override
    public void doGet(final HttpServletRequest request,
                      final HttpServletResponse response) throws ServletException,
                      IOException {
        final AsyncContext = request.startAsync();
        request.setAttribute("mainThread", Thread.currentThread().getName());
        AsyncContext.setTimeout(5000);
        AsyncContext.start(new Runnable() {
            @Override
            public void run() {
                // long-running task
                try {
                    Thread.sleep(3000);
                } catch (InterruptedException e) { . . . }
                request.setAttribute("workerThread", Thread.currentThread().getName());
                // Dispatch the request object
                AsyncContext.dispatch("/threadNames.jsp");
            }
        });
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 6

9

AsyncContext Class 5-10

- The code snippet shows the threadNames.jsp page.

```
<!DOCTYPE HTML>
<html>
<head>
<title>Asynchronous servlet</title>
</head>
<body>
    Main thread: ${mainThread}
    <br/>
    Worker thread: ${workerThread}
</body>
</html>
```

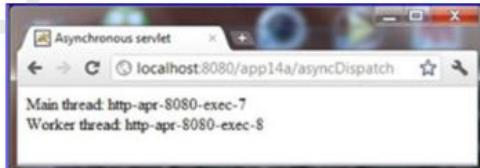
© Aptech Ltd.

Web Component Development Using Java/Session 6

10

AsyncContext Class 6-10

- Figure shows the name of the main thread and the name of the worker thread.



© Aptech Ltd.

Web Component Development Using Java/Session 6

11

AsyncContext Class 7-10

- The code snippet develops an asynchronous Servlet that sends progress updates to HTML page after every second.

```
import java.io.IOException;
import javax.servlet.AsyncContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class AsyncCompleteServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
                      IOException {
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        writer.println("<html><head><title>" + "Async Servlet</title></head>");
        writer.println("<body><div id='progress'></div>");
        final AsyncContext = request.startAsync();
        asyncContext.setTimeout(60000);
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 6

12

AsyncContext Class 8-10

```
asyncContext.start(new Runnable() {
    @Override
    public void run() {
        System.out.println("new thread:" + Thread.currentThread());
        for (int i = 0; i < 10; i++) {
            writer.println("<script>");
            writer.println("document.getElementById('" +
                           "progress').innerHTML = '" + i * 10) + "% complete'");
            writer.println("</script>");
            writer.println("writing . flush ()");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) { . . . }
        }
        writer.println("<script>");
        writer.println("document.getElementById('" +
                           "progress').innerHTML = 'DONE'");
        writer.println("</script>"); writer.println("</body></html>");
        asyncContext.complete();
    }
})
```

Contd.

© Aptech Ltd.

Web Component Development Using Java/Session 6

13

AsyncContext Class 9-10

- The code snippet displays the deployment descriptor, web.xml for configuring the AsyncCompleteServlet.

```
<web-app>
    .
    .
    <servlet>
        <servlet-name>AsyncComplete</servlet-name>
        <servlet-
        class>servlet.AsyncCompleteServlet</servlet-
        class>
        <async-supported>true</async-supported>
    </servlet>

    <servlet-mapping>
        <servlet-name>AsyncComplete</servlet-name>
        <url-pattern>/asyncComplete</url-pattern>
    </servlet-mapping>
</web-app>
```

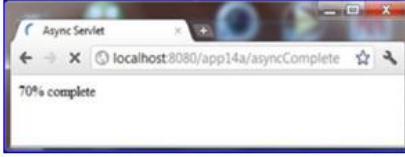
© Aptech Ltd.

Web Component Development Using Java/Session 6

14

AsyncContext Class 10-10

Figure shows the result of HTML page that receives progress updates from the asynchronous Servlet.



© Aptech Ltd., Web Component Development Using Java/Session 6. 15

Use slides 6 to 15 to explain AsyncContext class.

Tell them that `AsyncContext` class provide methods that can be used to obtain the instance of the `AsyncContext` within the `Servlet service()` method. The object of the `AsyncContext` class holds the state of the request and response objects in the servlet.

Then, using slide 6, explain the methods provided by the `AsyncContext` class.

Using slide 7, explain the steps to be performed to handle asynchronous request.

Using slides 8, explain the code for obtaining the asynchronous context within the `Servlet doGet()` or `doPost()` method.

Then, using slide 9 demonstrate the code that processes the request asynchronously in the Servlet. Tell them that the servlet supports asynchronous processing and its long-running task to keep sleep mode on for three seconds. Servlets `doGet()` method proves that the long-running task is executed in a different thread than the main thread.

The `doGet()` method attaches the name of the main thread and that of the worker thread to the `ServletRequest` and dispatches to a `threadNames.jsp` page. The `threadNames.jsp` page displays `mainThread` and `workerThread` thread names in the output.

Then, using slide 10, explain the `threadNames.jsp` page. It shows the name of the main thread and the name of the worker thread. Slide 11 displays the output of the code.

Then, consider another scenario where a servlet sends a progress update every second so that the user can monitor the progress. It sends HTML response and a simple JavaScript code to update the HTML page.

Using slides 12 to 14, explain the code that develops an asynchronous Servlet that sends progress updates to HTML page after every second. Tell them that in the code the `asyncContext` object is obtained by invoking the `startAsync()` on the request object. The object creates a thread that creates an HTML page and JavaScript code to update the progress element displayed with the value. Slide 15 displays the output of the code.

Slides 16 to 19

Let us understand AsyncListener Interface.

AsyncListener Interface 1-4

- ❖ The `AsyncListener` interface defines the methods that are used to notify the events occurring on the `AsyncContext` object.
- ❖ Table lists the methods of the `AsyncListener` interface.

Method	Description
<code>void onStartAsync(AsyncEvent event)</code>	It gets called when an asynchronous operation has been initiated.
<code>void onComplete(AsyncEvent event)</code>	It gets called when an asynchronous operation has completed.
<code>void onError(AsyncEvent event)</code>	It gets called in the event an asynchronous operation has failed.
<code>void onTimeout(AsyncEvent event)</code>	It gets called when an asynchronous has timed out.

© Aptech Ltd.

Web Component Development Using Java/Session 6

16

AsyncListener Interface 2-4

- ❖ The code snippet demonstrates the code that `AsyncListenerServlet` class that registers the listener to receive event notifications.

```
@WebServlet(name = "AsyncListenerServlet", urlPatterns = {"/asyncListener"}, asyncSupported = true)
public class AsyncListenerServlet extends HttpServlet {
    private static final long serialVersionUID = 62738L;
    @Override
    public void doGet(final HttpServletRequest request,
                     HttpServletResponse response) throws ServletException,
                     IOException {
        // Obtaining the Asynchronous context object
        final AsyncContext request.startAsync();
        //Setting the time out duration for asyncContext object
        asyncContext.setTimeout(5000);
        // Registering the events
        asyncContext.addListener(new MyAsyncListener());
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 6

17

AsyncListener Interface 3-4

```
//Obtaining a new thread
asyncContext.start(new Runnable() {
    @Override
    public void run() {
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) { ...}
        String greeting = "hi from listener";
        System.out.println("wait....");
        request.setAttribute("greeting", greeting);
        asyncContext.dispatch("/test.jsp");
    }
})
```

© Aptech Ltd.

Web Component Development Using Java/Session 6

18

AsyncListener Interface 4-4

❖ The code snippet shows the implementation of the AsyncListener interface methods.

```
public class MyAsyncListener implements AsyncListener {
    @Override
    public void onComplete(AsyncEvent asyncEvent)
        throws IOException {
        System.out.println("onComplete");
    }

    @Override
    public void onError(AsyncEvent asyncEvent)
        throws IOException {
        System.out.println("onError");
    }

    @Override
    public void onStartAsync(AsyncEvent asyncEvent)
        throws IOException {
        System.out.println("onStartAsync");
    }

    @Override
    public void onTimeout(AsyncEvent asyncEvent)
        throws IOException {
        System.out.println("onTimeout");
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 6 19

Use slides 16 to 19 to explain AsyncListener interface.

When the AsyncContext object is processing the request and response asynchronously, it passes through a series of events that monitor the life cycle of the AsyncContext object. To handle these events, the AsyncContext object can be registered with the AsyncListener interface. The AsyncListener interface defines the methods that are used to notify the events. Some of the methods provided by the AsyncListener interface are as follows:

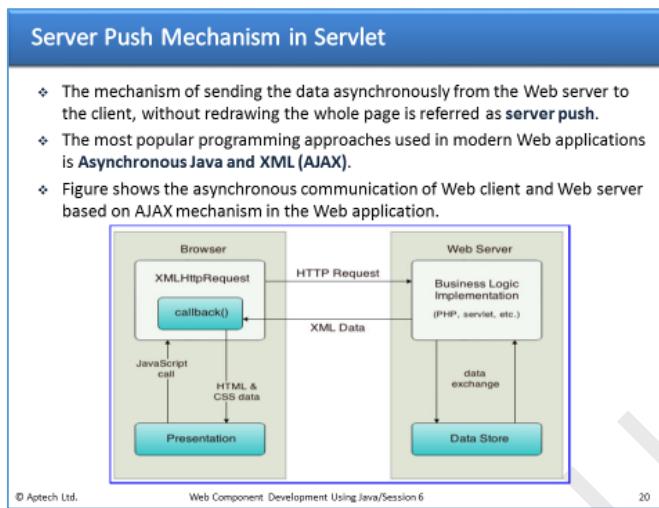
- void onComplete(AsyncEvent event) - notifies that an asynchronous operation has been completed.
- void onTimeout(AsyncEvent event) - notifies that an asynchronous operation has timed out.
- void onError(AsyncEvent event) - notifies that an asynchronous operation has failed to complete.
- void onStartAsync(AsyncEvent event) - notifies that a new asynchronous cycle is being initiated via a call to one of the ServletRequest.startAsync() methods.

All the four methods receive an AsyncEvent object that gets fired when the asynchronous operation is started, completion of asynchronous processing, error occurrence, and timeout.

Using slides 17 to 19, explain the codes showing AsyncListenerServlet class that registers the listener to receive event notifications and implements the AsyncListener interface methods.

Slide 20

Let us understand server push mechanism in servlet.



Use slide 20 to explain server push mechanism in servlet.

Tell them a more useful feature of servlet 3.0 is server push which is based on asynchronous communication. For example, a Google widget GTalk which allows the users to chat with each other. The main feature of GTalk application is that it does not approach or poll the server frequently to check for any new messages. Instead, whenever there is a new message on the server, it pushes the message to GTalk client asynchronously. This provides benefits such as no waste of resource and network bandwidth.

The asynchronous behavior of the Web technologies has led to the development of modern Web clients that allow the page to interact with the server and update the page without loading the whole page. The mechanism of sending the data asynchronously from the Web server to the client, without redrawing the whole page is referred as server push.

Tell them that there are many techniques to achieve server push mechanism in Web applications. However, one of the most popular programming approaches used in modern Web applications is Asynchronous Java and XML (AJAX).

Using AJAX, the client Web browser uses a JavaScript code that fetches a small amount of data from the Server and updates or integrates the displayed page with the received response from the server, without fetching the entire page.

Then, using the figure as shown on slide 20, explain asynchronous communication of Web client and Web server based on AJAX mechanism in the Web application. Tell them that the Web application uses JavaScript to modify the client's page, whereas the Web server uses the Java Servlet to process the HTTP request asynchronously.

The AJAX mechanism works with an XMLHttpRequest object that is used to pass the requests and responses asynchronously between the client and server. All modern Web browsers support the XMLHttpRequest object.

In-Class Question:

After you finish explaining Server Push, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which mechanism is used for sending the data asynchronously from the Web server to the client, without redrawing the whole page?

Answer:

Server Push

Slides 21 to 25

Let us understand developing asynchronous client.

Developing Asynchronous Client 1-5

- ❖ The XMLHttpRequest object is used in the JavaScript code to interact asynchronously with the Web server.
- ❖ The steps for creating the asynchronous JavaScript client are as follows:

Create an XMLHttpRequest object.

Send the request to the server using the methods of the XMLHttpRequest object.

Receive and process the response received from the server.

© Aptech Ltd. Web Component Development Using Java/Session 6 21

Developing Asynchronous Client 2-5

- ❖ The code snippet demonstrates how to check the support for XMLHttpRequest object in the HTML page and process the request and response using the XMLHttpRequest object.

```

<HTML>
<HEAD>
<SCRIPT TYPE="text/javascript">
  function getXMLHttpRequest() {
    var xmlhttpReq = false;

    // Creating XMLHttpRequest object for non-Microsoft browsers
    if (window.XMLHttpRequest) {
      xmlhttpReq = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
      try {
        // Creating XMLHttpRequest object in later versions
        // of Internet Explorer
        xmlhttpReq = new ActiveXObject("Msxml2.XMLHTTP");
      } catch (expl) {
    }
  }
</SCRIPT>

```

© Aptech Ltd. Web Component Development Using Java/Session 6 22

Developing Asynchronous Client 3-5

```

try {
  // to create XMLHttpRequest object in older versions
  // of Internet Explorer
  xmlhttpReq = new ActiveXObject("Microsoft.XMLHTTP");
} catch (exp2) {
  xmlhttpReq = false;    } //end of inner-catch
}

return xmlhttpReq;
}

/* AJAX call starts with this function*/
function makeRequest() {
  var httpRequest = getXMLHttpRequest();
  httpRequest.onreadystatechange =
    getReadyStateHandler(httpRequest);
  httpRequest.open("GET", "helloWorld.do", true);
  httpRequest.send();
}

```

Contd.

© Aptech Ltd. Web Component Development Using Java/Session 6 23

Developing Asynchronous Client 4-5

```
/* Returns a function that waits for the state change in
XMLHttpRequest*/
function getReadyStateHandler(xmlHttpRequest) {

    // an anonymous function returned
    // it listens to the XMLHttpRequest instance
    return function() {
        if (xmlHttpRequest.readyState == 4) {
            if (xmlHttpRequest.status == 200) {
                document.getElementById("hello").innerHTML =
xmlHttpRequest.responseText;
            } else {
                alert("HTTP error " + xmlHttpRequest.status + ":" +
xmlHttpRequest.statusText);
            }
        }
    };
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 6

24

Developing Asynchronous Client 5-5

- ❖ To send the request, the two methods namely, `open()` and `send()` of `XMLHttpRequest` object are invoked.
- ❖ These methods are used to prepare the HTTP Request and send it to the server.
- ❖ **Syntax:**

```
open(method,url,async)
```

where,

- **method:** Specifies the type of request method, that is, GET or POST is used to send the request.
- **url:** Specifies the location of the component handling the request on the server.
- **async:** Specifies whether the asynchronous communication is supported or not. The value can be true or false.

© Aptech Ltd.

Web Component Development Using Java/Session 6

25

Use slides 21 to 25 to explain developing asynchronous client.

Tell them the `XMLHttpRequest` object is used in the JavaScript code to interact asynchronously with the Web server. Following are the steps for creating the asynchronous JavaScript client:

- Create an `XMLHttpRequest` object.
- Send the request to the server using the methods of the `XMLHttpRequest` object. It contains a two methods namely, `open()` and `send()`. These methods are used to prepare the HTTP Request and send it to the server.
- Receive and process the response received from the server.

Open() method

The syntax of `open()` method is as follows:

```
open(method,url,async)
```

where,

- `method`: Specifies the type of request method, that is, GET or POST is used to send the request.
- `url`: Specifies the location of the component handling the request on the server.
- `async`: Specifies whether the asynchronous communication is supported or not. The value can be true or false.

send() method

The `send()` method sends the data to the request. In case of `GET` method, the data is sent as the query string, so nothing is sent with the `send()` method. However, if the method is `POST`, then `send()` method will have the data.

onreadystatechange event

Tell them that the JavaScript waits for the server response. Once the server is ready with the response, the `onreadystatechange` event is fired which indicates the change in the state. In the `onreadystatechange` event, you can specify what to do when the server response is ready to be processed. When `readyState` is 4 which means server has finished processing request and ready with the response, then check the status is 200, which means response status as OK.

On receiving the response, the callback function in the JavaScript is executed to display the response on the client's Web page.

Then, using slides 22 to 24, explain the code to check the support for `XMLHttpRequest` object in the HTML page and process the request and response using the `XMLHttpRequest` object. Tell them that the function `getXMLHttpRequest()` checks if the browser supports the `XMLHttpRequest` object. If the object is supported, then create an `XMLHttpRequest` object. For modern browser, the `XMLHttpRequest()` method is used, whereas for the older version of the Internet Explorer browser, the `XMLHttpRequest` object is obtained using `ActiveXObject()` method.

Using slide 25, explain the next step sends the request asynchronously using `open()` and `send()` methods to the server. Similarly, on receiving the response, the client JavaScript code executes the callback method to generate response.

Slides 26 to 28

Let us understand the implementation of non-blocking I/O support.

Implementation of Non-blocking I/O Support 1-3

- ❖ The non-blocking I/O is supported by:
 - javax.servlet.ServletInputStream
 - javax.servlet.ServletOutputStream
 - javax.servlet.ReadListener
 - javax.servlet.WriteListener
- ❖ The steps used for non-blocking I/O to process requests and writing responses inside service methods are as follows:
 - Enable the asynchronous mode for the Servlet using the @WebServlet annotation.
 - Obtain an input stream and/or an output stream from the request and response objects in the service() method.
 - Assign a read listener to the input stream and/or a write listener to the output stream.
 - Process the request and the response inside the listener's callback methods.

© Aptech Ltd. Web Component Development Using Java/Session 6 26

Implementation of Non-blocking I/O Support 2-3

- ❖ The code snippet demonstrates the use of non-blocking I/O in the asynchronous Servlet.

```
@WebServlet(urlPatterns={"/asyncioservlet"},  
asyncSupported=true)  
public class FirstServlet extends HttpServlet{  
    @Override  
    public void doPost(HttpServletRequest request,  
    HttpServletResponse response) throws IOException {  
        final AsyncContext acontextResponse = request.startAsync();  
        final ServletInputStream inputStream = request.  
        getInputStream();  
        inputStream.setReadListener(new ReadListener() {  
            byte buffer[] = new byte[4*1024];  
            StringBuilder sbuilder = new StringBuilder();  
            public void onDataAvailable() {  
                try {  
                    int length = inputStream.read(buffer);  
                    sbuilder.append(new String(buffer, 0, length));  
                } while(inputStream.isReady());  
            } catch (IOException ex) { ex.printStackTrace(); }  
        });  
        acontextResponse.getServletResponse().getWriter().write("...the  
        response...");  
        acontextResponse.complete();  
    }  
}
```

© Aptech Ltd. Web Component Development Using Java/Session 6 27

Implementation of Non-blocking I/O Support 3-3

```
try {  
    do {  
        int length = inputStream.read(buffer);  
        sbuilder.append(new String(buffer, 0, length));  
    } while(inputStream.isReady());  
} catch (IOException ex) { ex.printStackTrace(); }  
}  
public void onAllDataRead() {  
    try {  
        acontextResponse.getServletResponse().getWriter().write("...the  
        response...");  
    } catch (IOException ex) { ex.printStackTrace(); }  
    acontextResponse.complete();  
}  
}  
}
```

© Aptech Ltd. Web Component Development Using Java/Session 6 28

Use slides 26 to 28 to explain the implementation of non-blocking I/O support.

Tell them that non-blocking I/O provides the capability to recycle the idle threads by attaching them to the new requests, without consuming a new thread on each new request. This model is referred to as '**Thread per request**' which allows Web servers to handle growing users requests with a limited number of threads in the thread pool.

The non-blocking I/O is supported by the following classes in the servlet API:

`javax.servlet.ServletInputStream`, `javax.servlet.ServletOutputStream`,
`javax.servlet.ReadListener`, and `javax.servlet.WriteListener`.

Then, explain the steps that are used for non-blocking I/O to process requests and write responses inside service methods:

1. Enable the asynchronous mode for the Servlet using the `@WebServlet` annotation. For example, `@WebServlet(urlPatterns={"/asyncioservlet"}, asyncSupported=true)`
2. Obtain an input stream and/or an output stream from the request and response objects in the `service()` method.
3. Assign a read listener to the input stream and/or a write listener to the output stream.
4. Process the request and the response inside the listener's callback methods.

Then, using the slides 27 and 28, explain the use of non-blocking I/O in the asynchronous Servlet.

In-Class Question:

After you finish explaining implementation of non-blocking I/O support, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which mechanism executes application-specific blocking operations in a new thread, returning the thread associated with the request immediately to the container?

Answer:

Non-blocking I/O support

Slides 29 to 31

Let us understand protocol upgrade processing.

Protocol Upgrade Processing

- ❖ With Servlet 3.1 API, the HTTP protocol has been upgraded from HTTP 1.0 to HTTP 1.1.
- ❖ HTTP 1.0 was a stateless protocol.
- ❖ HTTP 1.1 can persist the connection between the client and the server.
 - The advantage of persistent connection is that the connection is alive and can be used for multiple requests with the client.

© Aptech Ltd. Web Component Development Using Java/Session 6 29

Upgrading to HTTP 1.1 Protocol 1-2

- ❖ In HTTP/1.1, on a current connection, clients can switch to a different protocol using the Upgrade header field.
- ❖ The code snippet demonstrates the use of headers to upgrade the protocol information in the response.

```
@WebServlet(urlPatterns={"/ABCDresource"})
public class ABCDUpgradeServlet extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) {
        if ("ABCD".equals(request.getHeader("Upgrade"))) {
            /* Accept upgrade request */
            response.setStatus(101);
            response.setHeader("Upgrade", "ABCD");
        } else { /* ... write error response ... */ }
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 6 30

Upgrading to HTTP 1.1 Protocol 2-2

- ❖ The code snippet shows the implementation of `HttpUpgradeHandler`.

```
package com.authentication;
import javax.servlet.ServletInputStream;
import javax.servlet.ServletOutputStream;
public class ABCDUpgradeHandler implements
HttpUpgradeHandler {
    public XYZPUpgradeHandler upgrade(XYZPUpgradeHandler
protocol){
        return protocol;
    }
    public void destroy() { }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 6 31

Use slides 29 to 31 to explain protocol upgrade processing.

With Servlet 3.1 API, the HTTP protocol has been upgraded from HTTP 1.0 to HTTP 1.1. HTTP 1.0 was a stateless protocol, whereas HTTP 1.1 can persist the connection between the client and the server. The advantage of persistent connection is that the connection is alive and can be used for multiple requests with the client.

Tell the students that in HTTP 1.1, clients can request to switch to a different protocol on the current connection by using the `Upgrade` header field. If the server accepts the request to switch to the protocol indicated by the client, it generates an HTTP response with status 101 (switching protocols). The client and the server communicate using the new protocol, after this exchange.

Use slides 30 and 31 to explain the use of headers to upgrade the protocol information in the response.

The following code snippet shows to read the request header and upgrade the protocol data.

```
if ("ABCD".equals(request.getHeader("Upgrade"))) {  
    /* Accept upgrade request */  
    response.setStatus(101);  
    response.setHeader("Upgrade", "ABCD");  
  
    ABCDUpgradeHandler handler =  
    request.upgrade(ABCDUpgradeHandler.class)  
  
} else { /* ... write error response ... */ }  
}
```

In-Class Question:

After you finish explaining upgrading to HTTP 1.1 protocol, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Name the field that can be used by client to switch to a different protocol.

Answer:

Upgrade

Slide 32

Let us summarize the session.

Summary

- ❖ The architecture of the Servlet is based on multithreaded model. In this model, the Web container normally creates a pool of threads ready to serve the client with the Servlet instance.
- ❖ The Java Servlet API provides asynchronous processing support for servlets and filters in the Web applications.
- ❖ In asynchronous processing, a Servlet thread waiting for a resource is released by the container and returned to the pool.
- ❖ The class `javax.servlet.AsyncContext` is used to process the request asynchronously within the Servlet.
- ❖ To handle these events, the `AsyncContext` object can be registered with the `AsyncListener` interface.
- ❖ The mechanism of sending the data asynchronously from the Web server to the client, without redrawing the whole page is referred as server push.
- ❖ One of the most popular programming approach used in modern Web applications to push the server data to the client is performed using AJAX mechanism.
- ❖ The AJAX mechanism works with an `XMLHttpRequest` object that is used to pass the requests and responses asynchronously between the client and server.
- ❖ Servlet API provides non-blocking I/O support for Servlets and filters to process input and output asynchronously in the request.
- ❖ With Servlet 3.1 API, the HTTP protocol has been upgraded from HTTP 1.0 to HTTP 1.1. HTTP 1.0 was a stateless protocol, whereas HTTP 1.1 can persist the connection between the client and the server.

© Aptech Ltd. Web Component Development Using Java/Session 6 32

In slide 32, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

6.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should explore the JavaServer Pages (JSP) technology to be taught in the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 7 – JavaServer Pages

7.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

7.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the need of JSP
- List the benefits of JSP
- Identify the situations where to use JSP and servlets
- Explain JSP architecture
- Identify various phases in the life cycle of a JSP page
- Explain the various scriptlet elements in JSP
- List and explain the use of various directives in JSP

7.1.2 Teaching Skills

To teach this session successfully, you should be aware of the basics of JavaServer Pages (JSP) and its features. Also, familiarize yourself with the life cycle of JSP page managed by the JSP container.

The session also covers the elements of a JSP page. To teach these concepts, familiarize yourself with the various scriptlet and directive tags available in a JSP page.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❖ Explain the need of JSP
- ❖ List the benefits of JSP
- ❖ Identify the situations where to use JSP and servlets
- ❖ Explain JSP architecture
- ❖ Identify various phases in the life cycle of a JSP page
- ❖ Explain the various scriptlet elements in JSP
- ❖ List and explain the use of various directives in JSP

Tell them that in this session, they will be introduced to the JavaServer Pages (JSP) and its features.

This session also provides an overview of the JSP architecture. Explain the two phases, that is, Translation phase and Execution phase of JSP life cycle. The session also discusses about the standard tags and custom tags available in JSP. Finally, it lists and explains the use of various directives in JSP.

7.2 In-Class Explanations

Slides 3 to 5

Let us understand the need of JSP with the help of a scenario.

Introduction 1-3

Scenario: A developer has to greet the user on his/her successful registration on the Web page.

❖ **Solution:**

- ❑ Can be accomplished using Servlet technology.
- ❑ Create a Servlet class named `GreetingServlet`.
- ❑ Processes the request and generates a response to be sent to the client.

© Aptech Limited. Web Component Development Using Java/Session 7 3

Introduction 2-3

❖ Figure shows the `GreetingServlet` class.

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

public class GreetingServlet extends HttpServlet {
    protected void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        String name = req.getParameter("name");
        String email = req.getParameter("email");
        String message = null;
        GregorianCalendar calendar = new GregorianCalendar();
        if (calendar.get(Calendar.AM_PM) == Calendar.AM)
            message = "Good Morning";
        else
            message = "Good Afternoon";
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<html>");
        out.println("<body>");
        out.println("<p>" + message + ", " + name + "</p>");
        out.println("<p> Thanks for registering your email(" + email
                  + ")</p>");
        out.println("<p> The Fro Java Team.</p>");
        out.println("</html>");
        out.println("</body>");
        out.close();
    }
}
```

HTML tags clubbed with Java code

© Aptech Limited. Web Component Development Using Java/Session 7 4

Introduction 3-3

❖ In the code:

- ❑ To generate the output, the `GreetingServlet` code uses Java code that will display the string value in the output.
- ❑ HTML elements embedded in the same Servlet that take care of the presentation of the string on the Web page.
- ❑ Embedding HTML elements along with the Java code result in the development of rigid applications, which have a tight coupling between presentation and application layer.
- ❑ Any modifications in the Servlet results in page re-compilation, which means embedded HTML elements are also re-compiled.

To segregate the presentation text from the application, Sun laid a new specification referred to as **JavaServer Pages (JSP)**.

© Aptech Limited. Web Component Development Using Java/Session 7 5

Use slides 3 to 5 to explain the need of JSP.

Begin the introduction by presenting a scenario where Servlet technology is used for generating dynamic response.

Consider a situation where a developer has to greet the user on his/her successful registration on the page. To accomplish this task, if the Servlet technology is used, the `GreetingServlet` class is created. The `GreetingServlet` processes the request and generates a response to be sent to the client.

Tell the students to observe that to generate the output, the `GreetingServlet` code uses Java code that will display the string value in the output. In addition to Java code, it also contains HTML elements embedded in the code that take care of the presentation of the string on the Web page. However, embedding HTML elements along with the Java code result in the development of rigid applications, which have a tight coupling between presentation and application layer. Thus, any modification in the Servlet will result in page re-compilation which means embedded HTML elements are also re-compiled.

Finally, tell them that to segregate the presentation text from the application, Sun Microsystems has laid a new specification referred to as JavaServer Pages (JSP). It is an extension of the existing Servlet API and leverages all the features of Servlets.

Tips:

JSP is similar to PHP. However, JSP pages are developed using Java programming language. A JSP page is translated into Servlet at runtime; this JSP page turned into a Servlet is cached and re-used until the contents of JSP page are modified.

Slide 6

Let us understand JSP page.

JSP Page

- ❖ The JSP specification is an extension of the existing Servlet API and leverages all the features of Servlets.
- ❖ JSP page:
 - ❑ A text document containing static content, JSP tags, and Java code which simplifies the generation of dynamic contents.
 - ❑ Static contents are expressed as HTML, XML, or XHTML markup tags and are used to generate the user interfaces on the page.
 - ❑ Java code and JSP elements are used to generate dynamic contents on the Web page.



© Aptech Limited. Web Component Development Using Java/Session 7 6

Use slide 6 to explain the JSP page. Tell the students that JSP is a technology used to develop interactive Web pages. JSP was developed by Sun Microsystems and is an improved version of Java servlets.

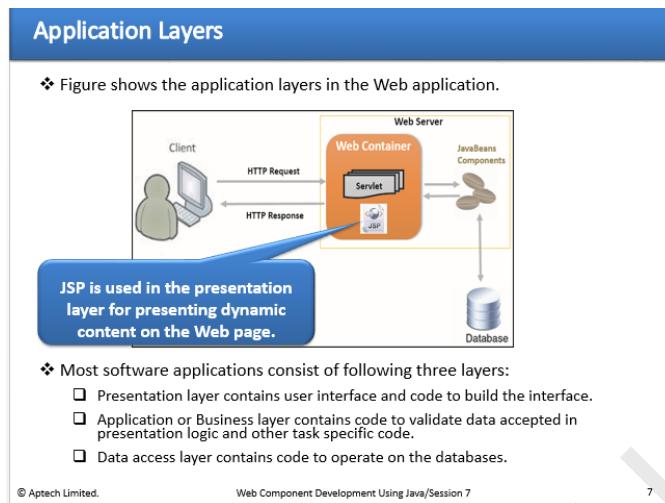
JSP is a technology that helps software developers create dynamically generated Web pages based on HTML, XML, or other document types. It allows developers to easily distribute application functionality to a wide range of page authors.

The contents on a JSP page which is developed as a text document contains static content, JSP tags, and Java code which simplifies the generation of dynamic contents.

The static contents are expressed as HTML, XML, or XHTML markup tags and are used to generate the user interfaces on the page. The Java code and JSP elements are used to generate dynamic contents on the Web page. Thus, the segregation of HTML tags and JSP tags separates the presentation code from the application code.

Slide 7

Let us understand application layers.



Use slide 7 to explain application layers. Tell the students that most software applications have a three-layered architecture comprised of presentation, business, and data layers.

The presentation layer usually contains UI and code to build the interface. The presentation layer is responsible for managing user interaction with the system and generally consists of components that provide a common bridge into the core business logic encapsulated in the business layer.

The business layer usually contains code to process data accepted from the user through presentation layer. This layer implements the core functionality of the system and encapsulates the relevant business logic.

The data layer usually contains code to operate on the databases. JSP is used in presentation layer for presenting dynamic content on the Web page.

Explain to the students that the figure on the slide that shows the application layers in the Web application.

In-Class Question:

After you finish explaining the application layers, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



In which layer of the three-tier architecture, is JSP used for presenting dynamic content on the Web page?

Answer:

JSP is used in presentation layer for presenting dynamic content on the Web page.

Slide 8

Let us understand the benefits of JSP.

Benefits of JSP

- Separation of Content Generation from Presentation**
 - JSP separates content generation from presentation by implementing presentation logic in a JSP page and content logic on server in a JavaBean component.
- Emphasizing Reusable Components**
 - JSP pages use reusable components, such as JavaBeans which can be used by multiple programs.
- Simplified Page Development**
 - JSP allows a Web developer with limited knowledge in scripting language to design a page.
- Access and Instantiate JavaBean Components**
 - JSP supports the use of JavaBean components with JSP language elements. The user can easily create and initialize beans and get and set their values.

© Aptech Limited. Web Component Development Using Java/Session 7 8

Use slide 8 to explain the benefits of JSP.

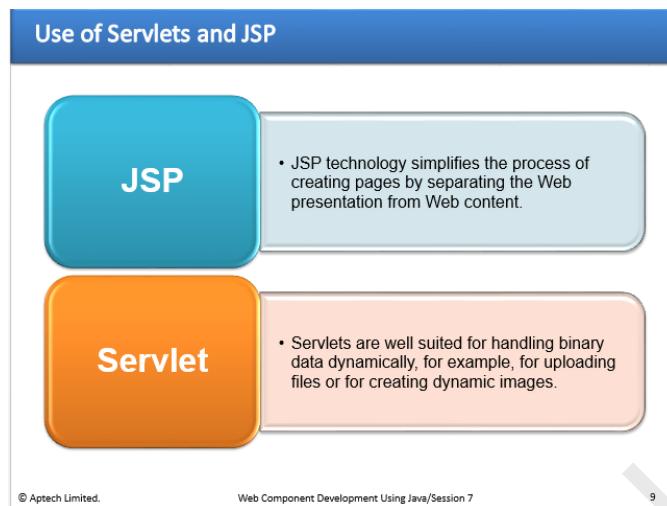
Tell the students that there are plenty of benefits of using JSP. JSP separates content generation from presentation by implementing presentation logic in a JSP page and content logic on server in a JavaBean component. For example, Web designers can alter and edit the static template portions of pages without disturbing the application logic. Developers can alter the logic at the component level without disturbing the corresponding pages that use the logic.

JSP pages use reusable components, such as JavaBeans which can be used by multiple programs. This is a portable, platform-independent component model that allows the developers to create components that can be reused anywhere. In JSP, JavaBeans contain business logic that returns data to a script on a JSP page and the returned data from the JavaBean component is formatted to be displayed by the browser. A JSP page uses a JavaBean component by setting and retrieving the properties that it carries.

JSP allows a Web developer with limited knowledge in scripting language to design a page. It supports the use of JavaBean components with JSP language elements. JSP allows the effortless distribution of the application functionality to the whole extent of page authors. These programmers can work on writing their HTML codes without bothering about writing Servlet code or learning Java programming language.

Slide 9

Let us understand the use of Servlets and JSP.



Use slide 9 to explain the use of Servlets and JSP.

Tell the students that JSP technology simplifies the process of creating pages by separating Web presentation from Web content. In many applications, the response sent to the client is a combination of template data and dynamically-generated data. In this situation, it is much easier to work with JSP pages than to do everything with servlets.

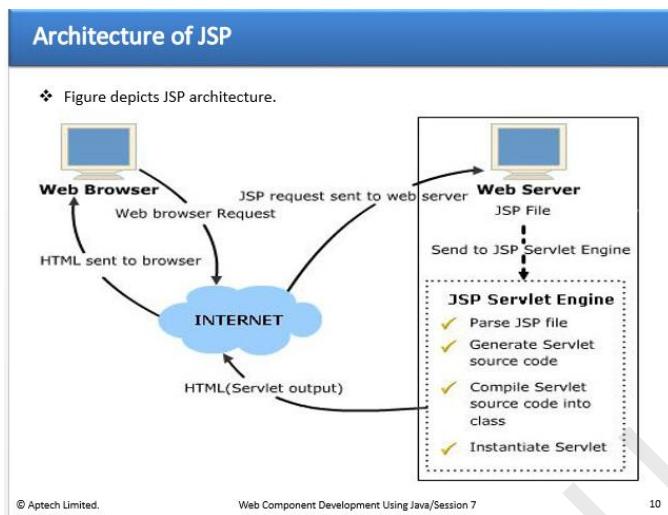
Tell them that JSPs can be used for most purposes, but there are some scenarios where Servlets are more appropriate. For example, Servlets are well-suited for handling binary data dynamically. For example, uploading files or creating dynamic images, since they need not contain any display logic.

Tips:

The difference between the Servlet and JSP technologies is not their life cycle or how container manages them at runtime. However, the difference between the two is the syntax that it offers for creating the same functionality. JSP pages are always simpler to create text-producing content. Servlets are best suited for sending raw data or bytes to a client or when the complete control is required on Java source code.

Slide 10

Let us understand the architecture of JSP.



Use slide 10 to explain the architecture of JSP.

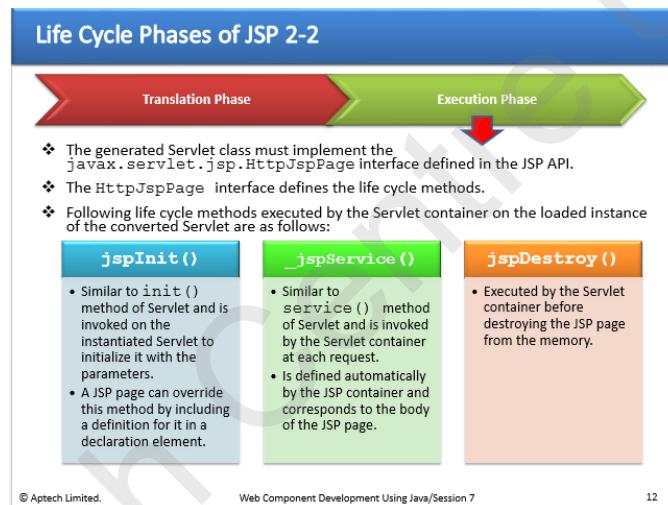
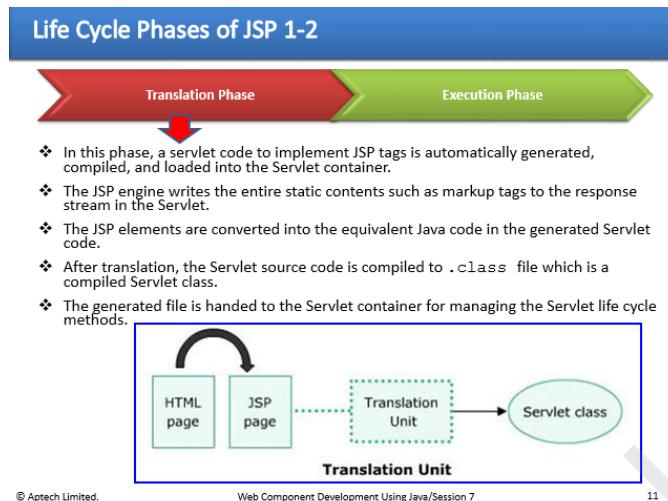
Tell the students that the figure depicts the JSP architecture.

Whenever the Web browser sends a URL to the Web server requesting a JSP page, the Web engine passes the requested JSP file to the JSP container which parses the JSP file to generate a Servlet source code. This servlet source code is compiled into a .class file.

The compiled .class file is sent to the Servlet container which instantiates the Servlet and manages the life cycle methods on the instance. The HTML output from the compiled Servlet is sent through the Internet and results are displayed on the user's Web browser.

Slides 11 and 12

Let us understand the life cycle phases of JSP.



Use slides 11 and 12 to explain the life cycle phases of JSP. Tell the students that during the process of parsing and compiling a JSP page into the Servlet, the JSP page is passed through two phases: **Translation Phase** and **Execution Phase**.

In the first phase, that is the translation phase, a servlet code to implement JSP tags is automatically generated, compiled, and loaded into the Servlet container. The JSP engine writes the entire static contents such as markup tags to the response stream in the Servlet. The JSP elements are converted into the equivalent Java code in the generated Servlet code. After translation, the Servlet source code is compiled to .class file which is a compiled Servlet class. The generated file is handed to the Servlet container for managing the Servlet life cycle methods.

Tell the students that the figure depicts the translation of JSP page into the Servlet class.

Then, discuss about the execution phase. Tell the students that the JSP life cycle's execution phase is almost similar to that of the Servlet life cycle, because ultimately, it's a servlet which is being executed. The Servlet class generated at the end of the translation phase represents the contract between container and the JSP page. According to the JSP specification, the servlet class must implement the `HttpJspPage` interface which defines the life cycle methods. JSP life cycle includes three methods `jspInit()`, `_jspService()`, and `jspDestroy()`.

Tell the students that `jspInit()`, `_jspService()`, and `jspDestroy()` are called the life cycle methods of the JSP.

jspInit() method is called immediately after the instance was created. It is called only once during JSP life cycle.

_jspService() method is called for every request of this JSP during its life cycle. It is defined automatically by the JSP container and corresponds to the body of the JSP page.

jspDestroy() method is called when this JSP is destroyed. With this call, the servlet serves its purpose and submits itself to heaven. This is the end of jsp life cycle.

In-Class Question:

After you finish explaining the life cycle phases of JSP, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



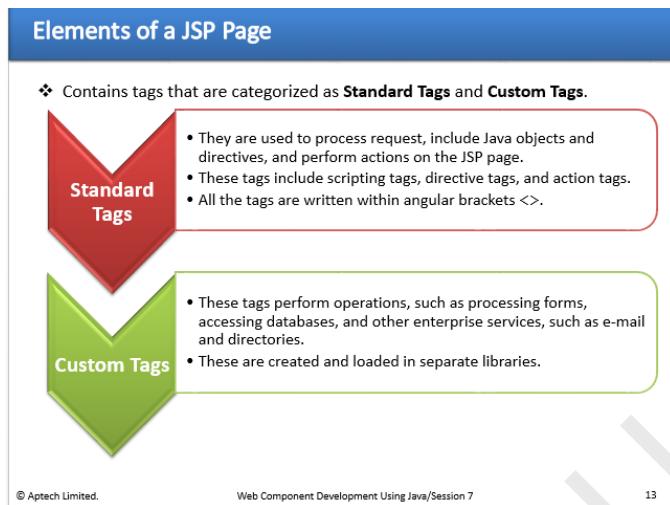
What are the life cycle methods in JSP?

Answer:

The life cycle methods in JSP are: `jspInit()`, `_jspService()`, and `jspDestroy()`.

Slide 13

Let us understand the elements of a JSP page.



Use slide 13 to explain the elements of a JSP Page.

Tell the students that a JSP page contains HTML tags, which display static content on a Web page. It also contains JSP tags which are used to generate dynamic content. The JSP tags are into standard tags and custom tags.

Next, explain the standard tags. Tell the students that standard tags are used to process request, include Java objects and directives, and perform actions on the JSP page. These tags include scripting tags, directive tags, and action tags.

Custom tags perform operations, such as processing forms, accessing databases, and other enterprise services, such as e-mail and directories.

Slides 14 to 18

Let us understand scripting tags.

Scripting Tags 1-5

- ❖ The JSP tags are used to embed Java code snippets in the JSP page for generating dynamic contents on the Web page. These tags are as follows:

JSP Expressions

- ❖ Can be used to display individual variables or the result of some calculation on a Web page.
- ❖ Contains a Java statement whose value will be evaluated and inserted into the generated Web page.
- ❖ Is evaluated and the result is converted into a string. This evaluated string will be displayed on the Web page.
- ❖ Syntax:

```
<%= Java Expression %>
```

where,

- <% = is the opening delimiter for the scriptlet.
- Java Expression indicates a valid Java statement.
- %> is the closing delimiter.

© Aptech Limited.

Web Component Development Using Java/Session 7

14

Scripting Tags 2-5

JSP Scriptlets

- ❖ Is used to embed Java code within an HTML code.
- ❖ Is inserted into the `_jspService()` method of the servlet.
- ❖ Are executed when the request of the client is being processed.
- ❖ Are used to add complex data to an HTML form.
- ❖ Syntax:

```
<% Java code fragment %>
```

- ❖ Figure depicts the use of scriptlet in a JSP page.

© Aptech Limited.

Web Component Development Using Java/Session 7

15

Scripting Tags 3-5

JSP Declarations

- ❖ Allow the user to define variables and methods that are inserted into the Servlet, outside the `service()` method.
- ❖ Can declare variables within the scriptlet tag also, but that variable will be local to that scriptlet.
- ❖ Variable declared using the declaration tag will be visible in the whole JSP page.
- ❖ Cannot be declared within the scriptlet tag.
- ❖ Can be used to define multiple variables of same data type.
- ❖ Syntax:

```
<%! Java declaration code %>
```

© Aptech Limited.

Web Component Development Using Java/Session 7

16

Scripting Tags 4-5

- The code snippet demonstrates the use of declaration tag in the JSP page along with its output.

```
<html>
...
<body>
<h3>Area of a Rectangle</h3>
<%! int length = 20, breadth = 30; area = 0; %>
<%! Private String units = "cm"; %>
<% area = length * breadth; %>
<p> The area of the Rectangle is:
<%= area %><%= units %>
</p>
</body>
</html>
```

© Aptech Limited.

Web Component Development Using Java/Session 7

17

Scripting Tags 5-5

Comments

- JSP comments are used for documenting JSP code which should not be visible to the client when viewing the source of the Web page.
- These comments are called server-side comments and are encoded within <%-- and --%> symbol.

Syntax:

```
<%-- a JSP comment --%>
```

- Example:** <%-- This is the JSP comment in a file --%>

© Aptech Limited.

Web Component Development Using Java/Session 7

18

Use slides 14 to 18 to explain scripting tags.

Tell the students that the JSP tags are used to embed Java code snippets in the JSP page for generating dynamic contents on the Web page. These tags include: Expressions, Scriptlets, Declarations, and Comments.

JSP expressions can be used to display individual variables or the result of some calculation on a Web page. It contains a Java statement whose value will be evaluated and inserted into the generated Web page. At run time, JSP expression is evaluated and the result is converted into a string. This evaluated string will be displayed on the Web page.

The syntax for JSP expressions is as follows:

```
<%= Java Expression %>
```

where,

<% is the opening delimiter for the scriptlet.

Java Expression indicates a valid Java statement and %> is the closing delimiter.

The example of JSP expression tag, displaying a welcome message is as follows:

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

Next, explain the JSP scriptlet tag. Tell the students that a JSP scriptlet is used to embed Java code within an HTML code. The Java code is inserted into the `_jspService()` method of the servlet. The scriptlets are executed when the request of the client is being processed. The scriptlets are used to add complex data to an HTML form.

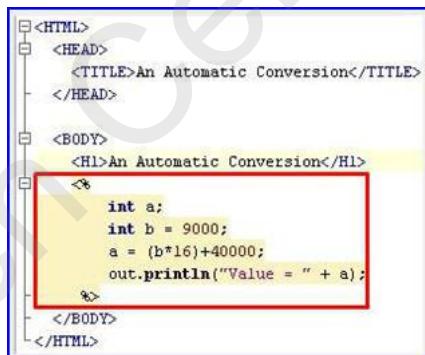
The syntax for JSP scriptlet tag is as follows:

```
<%= Java code fragment %>
```

The example of jsp scriptlet tag, displaying a welcome message is as follows:

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Tell the students that the following figure depicts the use of scriptlet in a JSP page.



Then, discuss about the JSP declaration tag. Tell the students that JSP declarations allow the user to define variables and methods that are inserted into the Servlet. The code written inside the JSP declaration tag is placed outside the `service()` method of auto generated servlet. So, it doesn't get memory at each request.

Also, tell them that a single declaration tag can be used to define multiple variables of same data type.

The syntax for JSP declaration tag is as follows:

```
<%! Java declaration code %>
```

Tell the students that the code snippet on slide 17 shows the use of declaration tag in the JSP page.

Finally, explain the JSP comments. Tell the students that JSP comments are used for documenting JSP code which should not be visible to the client when viewing the source of the Web page. These comments are called server-side comments and are encoded within <%-- and --%> symbol.

The syntax for JSP comments is as follows:

```
<%-- a JSP comment --%>
```

This JSP comment tag tells the JSP container to ignore the comment part from compilation. That is, the commented part of source code is not considered for the content parsed for 'response'.

The example of a JSP comment is as follows:

```
<%-- This is the JSP comment in a file --%>
```

Tips:

JSP scripting elements allows the user to use Java programming language statements in your JSP pages. Scripting elements are typically used to create and access objects, define methods, and manage the flow of control. Many tasks that require the use of scripts can be eliminated by using custom tag libraries, in particular the JSP Standard Tag Library (JSTL). Since one of the goals of JSP technology is to separate static data from the code needed to dynamically generate content, very sparing use of JSP scripting is recommended. Nevertheless, there may be some circumstances that require its use.

There are three ways to create and use objects in scripting elements:

- Instance and class variables of the JSP page's servlet class are created in declarations and accessed in scriptlets and expressions.
- Local variables of the JSP page's servlet class are created and used in scriptlets and expressions.
- Attributes of scope objects are created and used in scriptlets and expressions.

In-Class Question:

After you finish explaining scripting tags, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What is the syntax for JSP expressions?

Answer:

```
<%= Java Expression %>
```

Slides 19 and 20

Let us understand directive tags.

Directive Tags 1-2

- ❖ JSP directives control the processing of the entire page.
- ❖ These directives identify the packages to be imported and the interfaces to be implemented.
- ❖ These directives do not produce any output. They inform the JSP engine about the actions to be performed on the JSP page.
- ❖ These directives affect the overall structure of the Servlet class.
- ❖ **Syntax:**

```
<%@ directiveName attribute = "value"%>
```

where,

- ❑ directiveName is the name of the directive.
- ❑ attribute is attribute of directive.
- ❑ value is the value of attribute specified.

User can declare these directives anywhere on the page. However, mostly they are declared at the top of the JSP page.

© Aptech Limited.

Web Component Development Using Java/Session 7

19

Directive Tags 2-2

- ❖ In JSP, there are three directives:

- The page directive provides instructions that control the structure of the page
- The include directive includes static contents on the current JSP page
- The taglib directive includes the custom library tags on the JSP page

© Aptech Limited.

Web Component Development Using Java/Session 7

20

Use slides 19 and 20 to explain directive tags.

Tell the students that JSP directives provide directions and instructions to the container, telling it how to handle certain aspects of JSP processing. These directives identify the packages to be imported and the interfaces to be implemented. However, these directives do not produce any output. They inform the JSP engine about the actions to be performed on the JSP page.

Basically, a JSP directive affects the overall structure of the servlet class.

The syntax for JSP directive is as follows:

```
<%@ directiveName attribute = "value"%>
```

where,

directiveName is the name of the directive.

attribute is the attribute of directive.

value is the value of attribute specified.

A user can declare these directives anywhere on the page. However, mostly they are declared at the top of the JSP page.

Tell the students that there are three types of directive tags: page, include, and taglib.

The `page` directive provides instructions that control the structure of the page.

The `include` directive includes static contents on the current JSP page.

The `taglib` directive includes the custom library tags on the JSP page.

Tips:

Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.

The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

Slides 21 to 24

Let us understand the page directive.

page Directive 1-4

- ❖ Can be used to import classes from the packages, set the content type for the page, enable or disable page as error page, set the language on the page, and so on.
- ❖ Defines and manipulates a number of important attributes that affect the entire JSP page.
- ❖ Is written at the beginning of a JSP page.
- ❖ Can contain any number of page directives.
- ❖ All directives in the page are processed together during translation and result is applied together to the JSP page.

Example:

```
<%@ page import="java.util.* , java.io.* %>
```

- ❖ Used to import more than one package in the JSP page. If more than one package is imported, then separate the package names by commas.

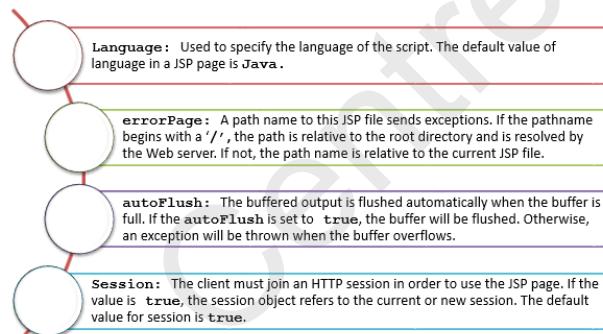
© Aptech Limited.

Web Component Development Using Java/Session 7

21

page Directive 2-4

- ❖ Some of the attributes that can be defined for a page are as follows:



© Aptech Limited.

Web Component Development Using Java/Session 7

22

page Directive 3-4

- ❖ Syntax:

```
<%@ page attribute %>
```

where,

- page** indicates page directive.
- attribute** specifies attribute-value pair of page directive.

- ❖ The valid parameters are as follows:

```
<%@ page language="java"
    extends="className"
    import="ClassName{,+}"
    session="true|false"
    buffer="none|sizeInKB"
    autoFlush="true|false"
    isThreadSafe="true|false"
    info="text"
    errorPage="jspUrl"
    isErrorPage="true|false"
    contentType="mimeType{;charset=charset}" %>
```

© Aptech Limited.

Web Component Development Using Java/Session 7

23

page Directive 4-4

❖ The code snippet demonstrates how to set the language on the JSP page by using the page directive.

```
<html>
<%@ page language="Java" %>
<head>
    <title>Testing Page Directive</title>
</head>
<body>
    <h1>Testing Page Directive</h1>
    This page is testing Page Directive.
</body>
</html>
```

Use slides 21 to 24 to explain page directive. Tell the students that the page directive is used to provide instructions to the container that pertain to the current JSP page.

It can be used to import classes from the packages, set the content type for the page, enable or disable page as error page, set the language on the page, and so on.

The page directive defines and manipulates a number of important attributes that affect the entire JSP page. Tell the students that page directives can be coded anywhere in the JSP page. However, by convention, page directives are coded at the top of the JSP page.

A JSP page can contain any number of page directives. All directives in the page are processed together during translation and result is applied together to the JSP page.

For example, `<%@ page import="java.util.* , java.io.*" %>` is used to import more than one package in the JSP page. If more than one package is imported, then separate the package names by commas.

Now, explain various attributes that can be defined for a page. Tell the students that there are various attributes associated with the page directive. These include: **language**, **errorCode**, **autoFlush**, and **session**.

language: The **language** attribute of the page directive is used to specify the language of the script. The default value of **language** in a JSP page is Java.

errorCode: The **errorCode** attribute tells the JSP engine which page to display if there is an error while the current page runs. The value of the **errorCode** attribute is a relative URL. If the pathname begins with a '/', the path is relative to the root directory and is resolved by the Web server. If not, the path name is relative to the current JSP file.

autoFlush: The **autoFlush** attribute specifies whether buffered output should be flushed automatically when the buffer is filled, or whether an exception should be raised to indicate buffer overflow. If the **autoFlush** is set to true, which is default value, the buffer will be flushed. Otherwise, an exception will be thrown when the buffer overflows.

Following directive causes the servlet to throw an exception when the servlet's output buffer is full:

```
<%@ page autoFlush="false" %>
```

This following directive causes the servlet to flush the output buffer when full:

```
<%@ page autoFlush="true" %>
```

Usually, the buffer and autoFlush attributes are coded on a single page directive as follows:

```
<%@ page buffer="16kb" autoflush="true" %>
```

Session: The session attribute indicates whether or not the JSP page uses HTTP sessions.

The client must join an HTTP session in order to use the JSP page. If the value is true, the session object refers to the current or new session. The default value for session is true.

The syntax for session attribute is as follows:

```
<%@ page attribute %>
```

where,

page indicates page directive.

attribute specifies attribute-value pair of page directive.

The valid parameters are as follows:

```
<%@ page language="java"
    extends="className"
    import="className{,+}"
    session="true|false"
    buffer="none|sizeInKB"
    autoFlush="true|false"
    isThreadSafe="true|false"
    info="text"
    errorPage="jspUrl"
    isErrorPage="true|false"
    contentType="mimeType{; charset=chars et}"%>
```

Tell the students that code snippet on slide 24 demonstrates how to set the language on the JSP page by using the page directive.

In-Class Question:

After you finish explaining page directive, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which attribute of the page directive is used to specify the language of the script?

Answer:

language attribute.

Slides 25 and 26

Let us understand `include` directive.

include Directive 1-2

- ❖ Is used to insert content of another resource in a JSP page at run time.
- ❖ Content are included physically during page translation:
 - The resource or content can be a file in a text-based format, such as text, HTML, or JSP.
 - The remote resource should be given with the proper relative file path in the `include` directive.
- ❖ **Syntax:**

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

where,

□ `include` indicates the directive and `Filename` specifies the name of a file to include along with relative path.

© Aptech Limited. Web Component Development Using Java/Session 7 25

include Directive 2-2

- ❖ Figure depicts the `include` directive.

```
<html>
  <head><title>Test for include directive </title></head>
<body bgcolor="white">
<font color="red">
  Today's date is :
  <%@ include file="date.jsp" %>
</font>
</body>
</html>
```

© Aptech Limited. Web Component Development Using Java/Session 7 26

Use slides 25 and 26 to explain the `include` directive. Tell the students that the `include` directive is used to insert content of another resource in a JSP page at runtime. The contents are included physically during page translation. The resource or content can be a file in a text-based format, such as text, HTML, or JSP.

The remote resource should be given with the proper relative file path in the `include` directive.

Mention that the `include` directives can be coded anywhere in the page. A good example of `include` directive is including a common header and footer with multiple pages of content.

The syntax for `include` directive is as follows:

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

where,

`include` indicates the directive.

`Filename` specifies the name of a file to include along with relative path.

Tell the students that the figure on slide 26 depicts the `include` directive.

Slides 27 and 28

Let us understand the `taglib` directive.

taglib Directive 1-2

- ❖ It allows the JSP page to create custom tags, which are defined by the user.
- ❖ A tag library is a group of custom tags that extend the functionality of a JSP page one after the other.
- ❖ The `taglib` directive specifies name of the tag library, which contains compiled Java code for a tag to be used.
- ❖ Using this tag library, the JSP engine determines what action is to be taken when a particular tag appears in a JSP page.
- ❖ **Syntax:**

```
<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>
```

where,

- ❑ `tagLibraryURI` specifies Uniform Resource Identifier (URI) that identifies the tag library descriptor.
- ❑ `prefix` specifies tag name that is used to define a custom tag.
- ❑ `tagPrefix` defines the prefix string in `prefix:tagname`.

© Aptech Limited.

Web Component Development Using Java/Session 7

27

taglib Directive 2-2

- ❖ Figure depicts the `taglib` directive.

```
<%@ taglib uri="tags" prefix="mt" %>
<HTML>
  <HEAD>
    <TITLE>Hello World</TITLE>
  </HEAD>
  <BODY BGCOLOR="#FFFFFF">
    <HR>
    <mt:helloWorld/>
    <HR>
  </BODY>
</HTML>
```

© Aptech Limited.

Web Component Development Using Java/Session 7

28

Use slides 27 and 28 to explain the `taglib` directive. Tell the students that the `taglib` directive allows the JSP page to create custom tags, which are defined by the user. Custom tags have access to all the objects that are available for a JSP page.

Mention that a tag library is a group of custom tags that extend the functionality of a JSP page one after the other. The `taglib` directive specifies name of the tag library which contains compiled Java code for a tag to be used. Using this tag library, the JSP engine determines what action is to be taken when a particular tag appears in a JSP page.

The syntax for `taglib` directive is as follows:

```
<%@ taglib uri="tagLibraryURI" prefix="tagPrefix" %>
```

where,

`tagLibraryURI` specifies Uniform Resource Identifier (URI) that identifies the tag library descriptor.

`prefix` specifies tag name that is used to define a custom tag.

`tagPrefix` defines the prefix string in `prefix:tagname`.

Tell the students that the figure on slide 28 depicts the `taglib` directive.

Slide 29

Let us summarize the session.

Summary

- ❖ JSP technology is used for developing dynamic Web sites and provides server-side scripting support for creating Web applications.
- ❖ A JSP page is a mixture of standard HTML tags, Web page content, and dynamic content that is specified using JSP elements.
- ❖ The JSP elements are expressions, scriptlets, comments, declarations, directives, and actions.
- ❖ Directives are used to specify the structure of the resulting servlet and actions are JSP tags that transfer control to other server objects or perform operations on other objects.
- ❖ The page directive is used to set the page attributes, such as the language to be used and encoding format.
- ❖ The include directive is used to insert a file referenced in the directive into the JSP page.
- ❖ The taglib directive links the JSP page to an XML document that describes a set of custom JSP tags and determines which tag handler class can implement the action of each tag.

© Aptech Limited. Web Component Development Using Java/Session 7 29

Using slide 29, summarize the session. End the session, with a brief summary of what has been taught in the session.

7.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the different types of implicit objects available in JSP and the use of implicit objects in a JSP page.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 8 – JSP Implicit Objects

8.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

8.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the concept of implicit objects in JSP
- List various types of implicit objects in JSP
- Explain how to use the request object
- Explain how to use the response object
- Explain how to use the out object
- Explain how to use the session object
- Explain how to use the application object
- Explain how to use the pageContext object
- Explain how to use the page object
- Explain how to use the config object
- Explain how to use the exception object

8.1.2 Teaching Skills

To teach this session successfully, you should be aware of what is the use of implicit objects in JSP. Also, familiarize yourself with the various types of implicit objects and how to work with them on the JSP page.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives
<ul style="list-style-type: none">❖ Explain the concept of implicit objects in JSP❖ List various types of implicit objects in JSP❖ Explain how to use the request object❖ Explain how to use the response object❖ Explain how to use the out object❖ Explain how to use the session object❖ Explain how to use the application object❖ Explain how to use the pageContext object❖ Explain how to use the page object❖ Explain how to use the config object❖ Explain how to use the exception object

Tell them in this session, they will be introduced to the concept of implicit objects in JSP. They will be learning a detailed explanation about the various types of implicit objects and their usage on the JSP page.

8.2 In-Class Explanations

Slide 3

Let us understand implicit objects.

Implicit Objects

- ❖ **Implicit objects:**
 - Are a set of Java objects that are available in every JSP page.
 - Are created and loaded by the Web container.
 - Are referred to as pre-defined variables.
 - Are accessible within the scripting elements in the JSP page.
- ❖ **Where are implicit objects created?**
 - When the JSP page is translated into a Servlet class, all the implicit objects declarations are taken within the `_jspService()` method.

© Aptech Limited. Web Component Development Using Java/Session 8 3

Use slide 3 to explain implicit objects. Begin by asking the students what they understand by JSP implicit objects. After listening to their answers, start explaining about implicit objects. Once the first two points are described, tell the students that implicit objects are Java objects that are automatically available on a JSP page, and thus, there is no need for Web programmers to create them.

Mention that when the JSP page is translated into a Servlet class, all the implicit objects declarations are taken within the `_jspService()` method. Implicit objects are also known as pre-defined variables and can be called directly without declaring them explicitly.

Tips:

JSP implicit objects are created by container while translating JSP page to Servlet source. The Web developers can use these objects directly in **scriptlets** that goes in service method of the Servlet. However, the Web developer can't use them in JSP Declaration because that code will go at class level.

Slides 4 and 5

Let us understand the types of implicit objects.

Types of Implicit Objects 1-2

- ❖ Some of the implicit objects provided by JSP are classified into following categories:
 - ❑ Input and Output Objects
 - ❑ Scope Communication and Control Objects
 - ❑ Servlet Objects
 - ❑ Error Objects

Input and Output Objects

- These include objects such as `request`, `response`, and `out`.
- The `request` object controls the data coming into the page.
- The `response` object controls the information generated as a result of `request` object.
- The `out` object controls the output data generated as a response to the request.

© Aptech Limited. Web Component Development Using Java/Session 8

4

Types of Implicit Objects 2-2

Scope Communication and Control Objects

- The scope communication objects provide access to all the objects available in the given scope.
- Objects in a JSP application are accessible according to the specified scope.
- The scope of an object is the section where that object is accessible.

Servlet Objects

- These objects provide information about the page context.
- These processes request object from a client and sends the response object back to the client.

Error Objects

- The error object handles error in a JSP page using an implicit object known as `Exception`.
- User can access this object by declaring JSP page as an error page.
- To do so, use the `isErrorPage` attribute of the `page` directive. For example, `<%@page isErrorPage="true" %>`.

© Aptech Limited. Web Component Development Using Java/Session 8

5

Use slides 4 and 5 to explain the classification of some of the implicit objects in different categories.

Tell them that JSP supports nine defined variables which are called as implicit objects. These objects are categorized into different categories. Then, list out the categories, explain each category in detail.

Following table shows the division of objects in the different categories:

Category	Implicit Objects
Input and Output objects	<code>request</code> - <code>javax.servlet.http.HttpServletRequest</code> , <code>response</code> - <code>javax.servlet.http.HttpServletResponse</code> , <code>out</code> - <code>javax.servlet.jsp.JspWriter</code>
Scope and Control objects	<code>session</code> - <code>javax.servlet.http.HttpSession</code> , <code>application</code> - <code>javax.servlet.ServletContext</code> , <code>page</code> - <code>java.lang.Object</code>
Servlet objects	<code>pageContext</code> - <code>javax.servlet.jsp.PageContext</code> , <code>config</code> - <code>javax.servlet.ServletConfig</code>
Error objects	<code>exception</code> - <code>javax.servlet.jsp.JspException</code>

In-Class Question:

After you finish explaining the types of implicit objects, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which category provides information about the page context?

Answer:

Servlet objects.

Slides 6 to 14

Let us understand input and output objects.

Input and Output Objects 1-9



request Object

- ❑ It contains information sent from the client browser through HTTP protocol.
- ❑ The information stored in the request object includes source of the request, requested URL headers, cookies, and parameters associated with the request.
- ❑ When the JSP page is translated into the Servlet class, the request object is passed as a reference of the `javax.servlet.http.HttpServletRequest` interface by the container.
- ❑ The scope of the request object is page-level which means that the object will be accessible only in the current page.

© Aptech Limited. Web Component Development Using Java/Session 8 6

Input and Output Objects 2-9

Table lists some of the methods of the `request object` available on the JSP page for the developers.

Method	Description
<code>java.lang.String getParameter(java.lang.String name)</code>	This method returns the value of a request parameter as a string, or <code>null</code> if the parameter does not exist.
<code>java.lang.String[] getParameterValues(java.lang.String name)</code>	This method returns an array of string objects which contains all the values given request parameter has, or <code>null</code> if the parameter does not exist.
<code>java.util.Enumeration getParameterNames()</code>	This method returns an enumeration of string objects, which contains the names of the parameters. If no parameters, this method returns an empty enumeration.
<code>void setAttribute(java.lang.String name, java.lang.Object o)</code>	This method stores an attribute in the request. Attributes are reset between requests.
<code>java.lang.Object getAttribute(java.lang.String name)</code>	This method returns the name of the string from the named attribute.
<code>java.lang.String getRemoteHost()</code>	This method returns the name of the client or the last proxy that sends the request. If the hostname is not resolved, this method returns the dotted-string form of the IP address.
<code>java.lang.String getRemoteAddr()</code>	This method returns the Internet protocol address of the client or last proxy that sent the request.

© Aptech Limited. Web Component Development Using Java/Session 8 7

Input and Output Objects 3-9

The code snippet demonstrates the methods of `request object` to receive data from a registration page.

```
<!-- index.jsp -->
.
.
<form action="register.jsp">
First Name:
<input type="text" name="firstname">
<br/> <br/>
Favorite Game:
<input type="checkbox" name="game" value="Cricket">Cricket<BR>
<input type="checkbox" name="game" value="Hockey">Hockey<BR>
<input type="checkbox" name="game" value="Baseball">Baseball<BR>
<input type="checkbox" name="game" value="Badminton">Badminton<BR>
<br/> <br/>
<input type="submit" value="Submit" name="Submit">
</form>
.
.
```

© Aptech Limited. Web Component Development Using Java/Session 8 8

Input and Output Objects 4-9

```
<!-- register.jsp -->
<%
    // Retrieves data from text field
    String firstName = request.getParameter("firstname");
    // Retrieves data from check boxes and list box
    String favGame[] = request.getParameterValues("game");
    // Displays the form data
    out.println("Name is: " + "<font color='blue'>" + firstName +
    "</font>");
    out.println("<br/><br/>Favorite Game: " + "\n");
    for (int i = 0; i < favGame.length; i++) {
        out.println("<t<font color='blue'>" + favGame[i] +
        "</font>" + "<br/><br/>The parameters sent to the Servlet are: <br/>
<font color='blue'>
<%
    Enumeration enumParam = request.getParameterNames();
    while (enumParam.hasMoreElements()) {
        String str = (String) enumParam.nextElement();
        out.println(str);
    }
<%>
</font>
    . . .

```

© Aptech Limited.

Web Component Development Using Java/Session 8

9

Input and Output Objects 5-9

response Object

- ❑ Manages the response generated by JSP and uses HTTP protocol to send the response to client.
- ❑ Is a reference of the javax.servlet.http.HttpServletResponse interface.

Table lists the methods of response object to be used in the JSP page.

Method	Description
public void addCookie(Cookie cookie)	The method adds the specified cookie to the response. This method can be called more than once to set more than one cookies.
public void sendRedirect(java.lang.String location) throws java.io.IOException	The method sends a temporary redirect response to the client using the specified redirect location URL. This method can also accept relative URLs.
public java.lang.String encodeURL(java.lang.String url)	The method is used to encode the specified URL by including sessionid. This form of the url can be used in html tags that use a url, such as <code><a href.....></code> . This enables the server to keep track of the session.
public java.lang.String encodeRedirectURL(java.lang.String url)	The method returns a rewritten url that can be used with the <code>sendRedirect</code> method of <code>response</code> object.

© Aptech Limited.

Web Component Development Using Java/Session 8

10

Input and Output Objects 6-9

out Object

- ❑ Represents the output stream, which will be sent to the client as a response for the request.
- ❑ Is an instance of the javax.servlet.jsp.JspWriter class.
- ❑ Uses all standard `write()`, `print()`, and `println()` methods defined in `javax.servlet.jsp.JspWriter` class to display the output.
- ❑ Has page scope.

© Aptech Limited.

Web Component Development Using Java/Session 8

11

Input and Output Objects 7-9

- ❖ Some of the methods of `out` object are as follows:
 - `void flush() throws java.io.IOException`
 - Flushes the buffer.
 - Only the invocation of `flush()` will flush all the buffers in a chain of writers and output streams.
 - May be invoked indirectly if the buffer size is more.
 - The code snippet shows how to flush the page content.


```
out.flush();
<jsp:forward page="login.jsp"/>
```
 - `void clear() throws java.io.IOException`
 - Clears the contents of the buffer.
 - If the buffer has been already been flushed, then the clear operation will throw an `IOException`.

© Aptech Limited.

Web Component Development Using Java/Session 8

12

Input and Output Objects 8-9

The code snippet shows how to clear the buffer.

```
// If buffer is not empty, buffer is cleared
if (out.getBufferSize() != 0)
    out.clear();

▫ void close() throws java.io.IOException - For the JspWriter, this method need not be invoked explicitly as the code generated by the JSP container will automatically include close()
```

The code snippet shows the use of `close()` method.

```
out.print("Welcome!!!");
out.close();
```

- `void println(java.lang.String x) throws java.io.IOException` - This method prints a string and then terminates the line. For example, `out.println("The output is:" + ex);`

© Aptech Limited.

Web Component Development Using Java/Session 8

13

Input and Output Objects 9-9

- `void print(java.lang.String s) throws java.io.IOException` - This method prints a string. If the argument is null, then the string 'null' is printed.

The code snippet demonstrates the use of `print()` method.

```
out.print("Welcome!!!");
out.print("To Aptech Training");
```

© Aptech Limited.

Web Component Development Using Java/Session 8

14

Use slides 6 to 14 to explain input and output objects. Tell the students that the input object represents the input data passed through an HTTP request to the JSP page, and the output object handles the content to be sent to the client in response.

Use slides 6 to 9 to describe the `request` object in detail. Mention that whenever a client requests a page, a fresh object that represents the request is created by the JSP engine.

The `request` object provides methods to get HTTP header information including form data, cookies, HTTP methods, and so on.

Explain the methods of the `request` object given in the table of slide 7 with the help of the following examples:

- Example for `public java.lang.StringgetParameter(java.lang.String name)` method: `value = request.getParameter("paramName");`
- Example for `public java.lang.String[]getParameterValues(java.lang.String name)` method: `String values[] = request.getParameterValues("paramName");`
- Example for `public java.util.EnumerationgetParameterNames()` method: `enumeration names = request.getParameterNames();`
- Example for `public voidsetAttribute(java.lang.String name, java.lang.Object o)` method:

```
String personName = new String("Grace");
request.setAttribute("name", personName);
```
- Example for `public java.lang.ObjectgetAttribute(java.lang.String name)` method: `String personName = (String)
request.getAttribute("name");`
- Example for `public java.lang.StringgetRemoteHost()` method: `host:
<%=request.getRemoteHost()%>`
- Example for `public java.lang.StringgetRemoteAddr()` method: `addr:
<%=request.getRemoteAddr()%>`

Use the code snippet in slides 8 and 9 to demonstrate the methods of `request` object to receive data from a registration page. Tell them that in the code, the `getParameter()` method is used to retrieve a value from the text field which is a string value. To retrieve multiple selected values from the check box, `getParameterValues()` method is used. This method returns multiple value in an array.

Mention that the `getParameterNames()` method is used to retrieve the names of the parameters sent in the request string.

Use slide 10 to explain `response` object. Tell them that whenever the server creates the `request` object, a `response` object is also created. In addition, it also describes the interfaces that are involved in the generation of new HTTP headers. A developer with the aid of this object can add HTTP status codes, new date stamps, new cookies, and so on.

Explain each method of the response object given in the table with the help of the following examples:

- Example for public void addCookie(Cookie cookie) method:

```
Cookie MyCookie = newCookie("RollNumber","156");
MyCookie.setMaxAge(60*60*24*7*26);
response.addCookie(MyCookie);
```
- Example for public void sendRedirect(java.lang.String location) throws java.io.IOException method:

```
response.sendRedirect("myserver.com/thePage.htm?ID=737");
```
- Example for public java.lang.String encodeURL(java.lang.String url) method:

```
response.encodeURL("buyerPage.jsp");
```
- Example for public java.lang.String encodeRedirectURL(java.lang.String url) method:

```
if (name == null) {
    response.sendRedirect(response.encodeURL("homePage.jsp"))}
```

Use slides 11 to 14 to explain in detail `out` object and some of its methods. Tell them that a response content is sent using this object. The original `JspWriter` object is flushed according to the buffering of the page. Using the `buffered='false'` attribute of the page directive, buffering can be turned off.

Use the code snippet in slide 12 to demonstrate how to flush the page content.

Use the code snippet in slide 13 demonstrate how to clear the buffer and the use of `close()` method.

Use the code snippet in slide 14 to demonstrate the use of `print()` method.

In-Class Question:

After you finish explaining Input and Output objects, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which input and output object is accessible only in the current page?

Answer:

Request object.

Slides 15 to 27

Let us understand scope communication objects.

Scope Communication Objects 1-13

- ❖ The scope communication objects include:

- ❖ **Session Object:**
 - ❑ Is an instance of `javax.servlet.http.HttpSession` interface.
 - ❑ Has session scope.
- ❖ Some of the methods of **session** object are:
 - ❑ `public boolean isNew()` - The `isNew()` method is used for checking whether the session is newly created in this page or not. It will return false, if a session already exists and true, otherwise.

Scope Communication Objects 2-13

- ❑ `public void setAttribute(java.lang.String name, java.lang.Object value)` - This method holds the objects in the existing session.

The code snippet demonstrates how to associate a new session object with the request.

```

<%>
if (session.isNew())
{
    UserLogin userLoginObj = new UserLogin(name,
    password);
    session.setAttribute("loginuser", userLoginObj);
}
%>
...

```

Scope Communication Objects 3-13

- ❑ `public java.lang.Object getAttribute(java.lang.String name)` - The method returns the object bound with the specified name in the current session, otherwise returns null.

The code snippet shows how to access the named session object.

```

...
<%
String name = (String)
session.getAttribute("loginuser");
%>


|                                                                    |                              |
|--------------------------------------------------------------------|------------------------------|
| User name: </td> <td>&lt;% out.print(name); %&gt; &lt;/td&gt;</td> | <% out.print(name); %> </td> |
|--------------------------------------------------------------------|------------------------------|


...

```

Scope Communication Objects 4-13

- **public java.util.Enumeration getAttributeNames ()** - The method returns an enumeration of string objects. The string contains the names of all the objects bound to this session.

- **public void invalidate ()** - The method invalidates a session and then, releases the objects bound to it.

The code snippet shows how to invalidate a session.

```
<%>
// Retrieve the user's session
HttpSession session = request.getSession();
// Invalidate the session
session.invalidate();
%>
```

- **public void removeAttribute(java.lang.String name)** - The method removes the object bounded with the specified name from this session. For example, session.removeAttribute("loginuser");

Scope Communication Objects 5-13

❖ application Object:

- Is used to share the data between all the application pages.
- Can be accessed by any JSP present in the application.
- Is an instance of the javax.servlet.ServletContext interface.
- Has application scope.

Some of the methods of application object are:

- **public void setAttribute(java.lang.String name, java.lang.Object object)** - The method returns the servlet container attribute with the given name or null, if there is no attribute.

The code snippet shows how to set an attribute to be accessed in any JSP page in the application.

```
...
<%>
String Initialized = "yes";
application.setAttribute("init", Initialized);
%>
```

Scope Communication Objects 6-13

- **public java.lang.Object getAttribute(java.lang.String name)** - The method returns the servlet container attribute with the given name or null, if there is no attribute.
- **public java.util.Enumeration getAttributeNames ()** - The method returns an enumeration containing the attribute names available within this servlet context.

Scope Communication Objects 7-13

The code snippet demonstrates how to retrieve all application objects set in the application in the JSP page.

```
<%
String username="john";
String password="apt001";

application.setAttribute("username", username);
application.setAttribute("password", password);

// Retrieves all attributes
Enumeration enum=application.getAttributeNames();
// Print the attributes
while(enum.hasMoreElements()){
    String attrName=(String)enum.nextElement();
    out.println(attrName+"<BR>");
}
%>
```

© Aptech Limited.

Web Component Development Using Java/Session 8

21

Scope Communication Objects 8-13

- **public void removeAttribute(java.lang.String name)** - This method removes the attribute with the given name from the servlet context. For example,
application.removeAttribute("password");
- **public java.lang.String getServerInfo()** - This method returns the name and version of the servlet container on which the servlet is running. For example,out.println("Server Information:"+ application.getServerInfo());
- **public void log(java.lang.String msg)** - This method writes the specified message to a servlet log file, usually an event log. For example,application.log (Hello log Message);

© Aptech Limited.

Web Component Development Using Java/Session 8

22

Scope Communication Objects 9-13

- ❖ **pageContext Object:**
 - Allows user to access all the implicit objects defined in the page scope.
 - Is an instance of the `javax.servlet.jsp.PageContext class`.
 - Uses methods defined in the `PageContext class` to access implicit objects in a Web page.
- ❖ **PageContext class provides following fields that are used by pageContext object to find the scope or specify the scope of the objects:**

PAGE_SCOPE	• Specifies the scope for attributes stored in the <code>pageContext object</code> . This is the default.
REQUEST_SCOPE	• Specifies the request scope for attributes that remain available until current request is complete.
SESSION_SCOPE	• Specifies the scope for attributes stored in the <code>session object</code> .
APPLICATION_SCOPE	• Specifies the scope for attributes stored in the <code>application object</code> that remains available until it is reclaimed.

© Aptech Limited.

Web Component Development Using Java/Session 8

23

Scope Communication Objects 10-13

- ❖ The `pageContext` object provides the following methods to access all the attributes defined by implicit object in a page:
 - `void setAttribute(java.lang.String name, java.lang.Object value, int scope)` - This method registers the name and value specified with appropriate scope. If the value passed in is null, this method functions in the same way as `removeAttribute(name, scope)`.
 - `java.lang.Object getAttribute(java.lang.String name)` - This method returns the object associated with the name in the page scope, otherwise it returns null. It throws `java.lang.NullPointerException`, if the name is null.

© Aptech Limited.

Web Component Development Using Java/Session 8

24

Scope Communication Objects 11-13

The code snippet shows the `pageContext` object used to set and retrieve the object.

```
<%
String personName = "Cleveland";
pageContext.setAttribute("name", personName,
APPLICATION_SCOPE);
%>
...
<%
String personName = (String)
pageContext.getAttribute("name", APPLICATION_SCOPE);
out.print("Person name:" + personName);
%>
```

- `public abstract java.util.Enumeration getAttributeNamesInScope(int scope)` - This method enumerates all the attributes in a given scope. It throws `java.lang.IllegalArgumentException` if the scope is invalid.

© Aptech Limited.

Web Component Development Using Java/Session 8

25

Scope Communication Objects 12-13

The code snippet demonstrates how to retrieve all the objects with request scope.

```
// Prints all attribute names with request scope
<%
for (java.util.Enumeration e =
pageContext.getAttributeNamesInScope(javax.servlet.jsp.PageContext.REQUEST_SCOPE) ;
e.hasMoreElements(); ) { %
<%= e.nextElement() %><br>
<% } %>
```

- `public abstract void removeAttribute(java.lang.String name)` - This method removes the object reference associated with the given name from all the scopes. It throws `java.lang.NullPointerException`, if the name is null.

© Aptech Limited.

Web Component Development Using Java/Session 8

26

Scope Communication Objects 13-13

The code snippet demonstrates how to remove the objects from the JSP page.

```
<%  
out.println("User Name: " +  
request.getAttribute("Name") + "<br/>");  
out.println("Password: " +  
request.getAttribute("Password") + "<br/>");  
pageContext.removeAttribute("Password");  
%>
```

Use slides 15 to 27 to explain scope communication objects. Tell the students that in addition to the implicit objects, a JSP page can also access Java objects that are created explicitly. These objects are either created within the scriptlets or through actions. Example of an action can be creation of a JavaBean object on the Web page. Each created object is associated with a scope attribute which defines where the object is accessible and its lifetime.

Tell them that the session communication objects include, `session object`, `application object`, and `pageContext object`.

Use slides 15 to 18 to explain in detail `session object` and some of the methods of `session object`. Tell them that the `session object` provides all the objects available in the JSP pages within the current session. The requests of the client can be tracked using this object.

Ask them how a session terminates. After listening to their answer, tell them that a session expires when the user does not send any request for a long time or when the user closes the Web browser.

Explain each method of `session object` with an appropriate example. For `public boolean isNew ()` method, you can give them the example of a Web application such as payments page. Tell them that when a user opens a page, the server needs to check whether a session already exists or not.

`public void setAttribute(java.lang.String name, java.lang.Object value)` method, you can give them the example of the payments page, in which, when the user opens a page with the username, the `setAttribute ()` method holds the objects in the existing session.

Use the code snippet in slide 16 to demonstrate how to associate a new session object with the request. Explain that, in the code, the `userLoginObj` is an object of the `UserLogin` class which stores session data, such as `username` and `password` and the `setAttribute ()` method maps the `userLoginObj` to the `loginuser` which as a name to access the object.

Use slide 17 to explain `public java.lang.Object getAttribute(java.lang.String name)` method and to display the code snippet that shows how to access the named HttpSession object.

Use slide 18 to explain `public java.util.Enumeration getAttributeNames()` method, `public void invalidate()` method, and `public void removeAttribute(java.lang.String name)` method. Use the code snippet in slide 18 to demonstrate how to invalidate a session.

Tips:

Session tracking is enabled by default in JSPs, and a fresh HttpSession object is flushed for each new client automatically. Session tracking can be disabled by explicitly turning it off. This is done by setting the page directive `session` attribute to false as shown:

```
<%@ page session="false" %>
```

Use slides 19 to 22 to explain application object and some of the methods of application object. Tell the students that all users can share the information of a given application using the application object. Application object is created when the JSP page is initialized. This object gets deleted when the JSP page is deleted using the `jspDestroy()` method.

Tell them that `public void setAttribute(java.lang.String name, java.lang.Object object)` method binds an object to a given attribute name in this servlet context.

Use the code snippet in slide 19 to demonstrate how to set an attribute to be accessed in any JSP page in the application.

Use the code snippet in slide 21 to demonstrate how to retrieve all application objects set in the application.

Use slides 23 to 27 to explain `pageContext` object. The developer can use `pageContext` to get and set attributes with different scopes and to forward request to other resources. `pageContext` object also hold reference to other implicit object.

Use slide 23 to explain the fields provided by `pageContext` object. While explaining `SESSION_SCOPE` field, tell them that `SESSION_SCOPE` specifies the scope for attributes stored in the `session` object and it remains available until `HttpSession` object is invalidated.

Use slides 24 to 27 to explain the methods provided by `pageContext` object. Tell them that the `pageContext` object provides methods to access all attributes defined by implicit object in a page and it also provide methods to transfer the control from one Web page to another.

Explain the **Syntax** of `getAttribute()` method as :

```
public abstract java.lang.Object getAttribute(java.lang.String name)
```

where,

name is attribute name to be retrieved.

Following code snippet provides an example for the pageContext object:

```
<%-- pageContext object example --%>
<% pageContext.setAttribute("Test", "Test Value"); %>
<strong>PageContext attribute</strong>: {Name="Test",
Value="<%=pageContext.getAttribute("Test") %>" }<br><br>
```

Then, use the code snippet in slide 25 to demonstrate the pageContext object used to set and retrieve the object set in the application scope.

Use the code snippet in slide 26 to demonstrate how to retrieve all the objects with request scope.

Use the code snippet in slide 27 to demonstrate how to remove the objects from the JSP page.

Slides 28 to 32

Let us understand Servlet objects.

Servlet Objects 1-5

- ❖ Include objects:

```

graph LR
    A[page Object] --> B[config Object]
  
```

page Object:

- ❑ It is an instance of the `java.lang.Object`.
- ❑ It is an instance of the servlet processing the current request in a JSP page.
- ❑ It has page scope.

© Aptech Limited. Web Component Development Using Java/Session 8 28

Servlet Objects 2-5

The code snippet demonstrates how to access the methods of the Servlet through page object.

```

<%@ page %>
<%!
    out.println(this.getServletInfo());
    out.println(((Servlet)page).getServletInfo());
%>
  
```

- ❖ config Object:

- ❑ Stores the information of the servlet, which is created during the compilation of a JSP page.
- ❑ Is an instance of the `javax.servlet.ServletConfig` interface.
- ❑ Provides methods to retrieve servlet initialization parameters.
- ❑ Represents the configuration of the servlet data where a JSP page is compiled.
- ❑ Has page scope.

© Aptech Limited. Web Component Development Using Java/Session 8 29

Servlet Objects 3-5

- ❑ Some of the methods of config object are:

```

public java.lang.String getInitParameter(java.lang.String name)
  
```

- ❑ This method returns a string which contains the value of the named initialization parameter, otherwise it returns null.

```

public java.util.Enumeration getInitParameterNames()
  
```

- ❑ Returns the names of the servlet's initialization parameters as an enumeration of string objects.

© Aptech Limited. Web Component Development Using Java/Session 8 30

Servlet Objects 4-5

Code Snippet first retrieves all the init parameters `getInitParameterNames()` and then, prints each parameter name using `getInitParameter()`.

```
public void getParameters(ServletConfig config)
throws ServletException {
    // Retrieves all init parameters
    Enumeration params =
    config.getInitParameterNames();
    // Prints the init parameter names and values
    while (params.hasMoreElements()) {
        String name = (String) params.nextElement();
        System.out.println("Parameter Name: " + name +
        " Value: " + config.getInitParameter(name));
    }
}
```

© Aptech Limited.

Web Component Development Using Java/Session 8

31

Servlet Objects 5-5

`public ServletContext getServletContext()`

- ❑ This method returns a reference to the servlet context.
- ❑ The code snippet given shows how to set a JavaBean object in the `ServletContext`.

```
ServletContext context =
config.getServletContext();
context.setAttribute("dbBean", dbBean);
```

`public java.lang.String getServletName()`

- ❑ This method returns the name of this servlet instance.
- ❑ It can get the name that may be provided through the server administration or assigned in the Web application deployment descriptor.
- ❑ For example, `String servletName =
config.getServletName();`

© Aptech Limited.

Web Component Development Using Java/Session 8

32

Use slides 28 to 32 to explain Servlet objects. Tell the students that `Servlet` object is a representation of the JSP page and it includes `page` object and `config` object. Mention that a `page` object is an instance of the servlet processing the current request in a JSP page.

Use the code snippet in slide 29 to demonstrate how to access the methods of the Servlet through `page` object. Mention that as the variable `page` is of the type `Object`, to access the Servlet methods, you have to type cast it with the `Servlet`.

Use slide 29 to explain `config` object. This is a Servlet configuration object and is mainly used to access configuration information such as Servlet context, Servlet name, and the configuration data is stored in the form of initialization parameters. The `javax.servlet.ServletConfig` interface provides methods to retrieve Servlet initialization parameters configured in the deployment descriptor file.

Use slides 30 to 32 to explain the methods of `config` object. While explaining `public java.util.Enumeration getInitParameterNames()`, mention that this method returns an empty enumeration object, if the servlet has no initialization parameters.

Use the code snippet in slide 31 to demonstrate the method that retrieves all the `init` parameters using `getInitParameterNames()` and then prints each parameter name using `getInitParameter()`.

Use the code snippet in slide 32 to show how to set a JavaBean object in the `ServletContext`.

Following code snippet is an additional example that can be explained for fetching Servlet details using the `config` object:

`web.xml`

```
<web-app>
<servlet>
<servlet-name>FirstServlet</servlet-name>
<jsp-file>/index.jsp</jsp-file>
</servlet>

<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/index</url-pattern>
</servlet-mapping>
</web-app>
```

index.jsp page

```
<html>
<head> <title> Config Implicit Object</title>
</head>
<body>
<%
String sname=config.getServletName();
out.print("Servlet Name is: "+sname);
%>
</body>
</html>
```

The output of the code snippet is:

Servlet Name is: FirstServlet

In-Class Question:

After you finish explaining Servlet objects, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



What is the difference between page object and pageContext object?

Answer:

The page object is of `java.lang.Object` class and refers to the generated Servlet class, whereas `pageContext` object is of `javax.servlet.jsp.PageContext` class and is used for storing and retrieving page-related information.

Slides 33 and 34

Let us understand exception object.

exception Object 1-2

- ❖ Runtime errors can be processed using the exception object present in the JSP page.
- ❖ The exception object:
 - Is used to trace the exception thrown during execution.
 - Is available to the JSP page that is assigned as an error page.
 - Is the instance of `java.lang.Throwable` class.
 - Has page scope.
- ❖ Some of the methods of exception object are:
 - `public String getMessage()`**
 - Returns the descriptive error message associated with the exception when it was thrown.
 - `public void printStackTrace(PrintWriters)`**
 - Prints the execution stack in effect when the exception was thrown to the designated output stream.

© Aptech Limited.

Web Component Development Using Java/Session 8

33

exception Object 2-2

- `public String toString()`**
 - Returns a string combining the class name of the exception with its error message.
- Code Snippet shown demonstrates how to handle exceptions in the JSP.


```
<%@ page isErrorPage="true" %>
<html>
<head>
<title>Implicit Object</title>
</head>
<body>
<h1>Implicit Object: Exception</h1>
The following error has been detected :<br>
<b><%= exception %></b><br>
<% exception.printStackTrace(out); %>
</body>
</html>
```

 - The `isErrorPage` attribute is set to true, which indicates that the page is an error page. The JSP expression (`<%= %>`) is used to create an instance of the exception object.
 - The `out` object is passed as an argument to the `printStackTrace ()` method which will display the stack trace information.

© Aptech Limited.

Web Component Development Using Java/Session 8

34

Use slides 33 and 34 to explain exception object. Tell the students that while executing and processing client requests, runtime errors can occur either inside or outside the JSP page. These errors can be processed using the exception object present in the JSP page. When an error occurs, the execution of a JSP page is terminated.

The `exception` object is used to handle errors in a JSP page. This object is only available to the JSP pages, which has `isErrorPage` set to true.

Use the code snippet in slide 34 to demonstrate how to handle exceptions in the JSP.

Tips:

The exception object is available only in error pages. JSP provides an option to specify error Page for each JSP, and each time the page throws an exception, the error page is invoked by the JSP container automatically.

Following code snippet provides an additional case study where two numbers are accepted from the user and performed with division:

index.jsp page

```
<html>
<head>
<title> Division of Two Integers </title>
</head>
<body>
<form action="division.jsp">
    Input First Integer:<input type="text" name="firstnum" />
    Input Second Integer:<input type="text" name="secondnum" />
    <input type="submit" value="Get Results"/>
</form>
</body>
</html>
```

division.jsp

```
<%@ page errorPage="exception.jsp" %>
<%
String num1=request.getParameter("firstnum");
String num2=request.getParameter("secondnum");
int v1= Integer.parseInt(num1);
int v2= Integer.parseInt(num2);
int res= v1/v2;
out.print("Output is: "+ res);
%>
```

Here we have specified exception.jsp as errorPage which means if any exception occurs in this JSP page, the control will immediately transferred to the exception.jsp JSP page. The **errorPage attribute of Page Directive** is used to specify the exception handling JSP page.

exception.jsp

```
<%@ page isErrorPage="true" %>
Got this Exception: <%= exception %>
Please input the correct data.
```

JSP page we have set **isErrorPage** to true which is also an attribute of **Page** directive, used for making a page eligible for exception handling. Since this page is defined as an exception page in division.jsp, in case of any exception condition this page will be invoked.

Slide 35

Let us summarize the session.

Summary

- ❖ JSP implicit objects are a set of Java objects that are created and maintained automatically by the Web container.
- ❖ The request object represents the request from the client for a Web page.
- ❖ The response implicit object handles the response generated by JSP and sends response to the client.
- ❖ The out object represents the output stream.
- ❖ The Web server creates a session object for multiple requests sent by a single user. The application object represents the application to which the JSP page belongs.
- ❖ The pageContext object allows user to access all the implicit objects defined in the page scope.
- ❖ The page object provides access to all the objects, which are defined in a Web page.
- ❖ The config object stores the information of the servlet.
- ❖ When an error occurs, the execution of a JSP page is terminated. The exception implicit object is used to trace exceptions that occur in a JSP page.

© Aptech Limited. Web Component Development Using Java/Session 8 35

In slide 35, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

8.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the standard actions and JavaBeans that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 9 – Standard Actions and JavaBeans

9.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

9.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the concept of standard actions in JSP
- Explain how to use the <jsp:include> element
- Explain how to use the <jsp:forward> element
- Explain how to use the <jsp:param> element
- Explain how to use the <jsp:plugin> element
- Explain how to use the <jsp:fallback> element
- Explain how to use the <jsp:text> element
- Explain the concept of JavaBeans
- Explain how to declare and access JavaBeans components in JSP
- Explain accessing JavaBean properties from scripting elements
- Explain how to access non-string data type properties from scripting elements
- Explain how to access indexed properties from scripting elements

9.1.2 Teaching Skills

To teach this session successfully, you should be aware with concept of standard actions in JSP. You should familiarize yourself with the working of the various standard actions in JSP. The knowledge on the concept of JavaBean and its properties. You should be aware of accessing JavaBean properties from scripting elements and also accessing non-string data type properties from scripting elements. Finally, you should be able to explain accessing indexed properties from scripting elements.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives
<ul style="list-style-type: none">❖ Explain the concept of standard actions in JSP❖ Explain how to use the <jsp:include> element❖ Explain how to use the <jsp:forward> element❖ Explain how to use the <jsp:param> element❖ Explain how to use the <jsp:plugin> element❖ Explain how to use the <jsp:fallback> element❖ Explain how to use the <jsp:text> element❖ Explain the concept of JavaBeans❖ Explain how to declare and access JavaBeans components in JSP❖ Explain accessing JavaBean properties from scripting elements❖ Explain how to access non-string data type properties from scripting elements❖ Explain how to access indexed properties from scripting elements

Tell them that in this session, they will be introduced to the concept of standard actions in JSP. Discuss the various standard action elements and explain how they are used in JSP.

Explain the concept of java beans. Discuss how to declare and access JavaBeans components in JSP and accessing JavaBean properties from scripting elements and accessing non-string data type properties from scripting elements. Finally, explain accessing indexed properties from scripting elements.

9.2 In-Class Explanations

Slide 3

Let us understand standard actions.

The slide has a blue header bar with the title "Introduction". Below the header, there is a bulleted list under the heading "Standard actions:":

- Are XML like tags.
- Take the form of an XML tag with a name prefixed `jsp`.
- Are used for:

Below this list are four colored boxes, each containing a bullet point:

- Forwarding requests and performing includes in pages.
- Embedding the appropriate HTML on pages.
- Interacting between pages and JavaBeans.
- Providing additional functionality to tag libraries.

At the bottom left of the slide is the copyright notice "© Aptech Limited.", at the bottom center is "Web Component Development Using Java/Session 9", and at the bottom right is the number "3".

Use slide 3 to explain standard actions.

Tell the students that standard action are similar to XML tags. They are used to control the behavior of the Servlet engine. With help of standard actions, user can dynamically insert a file, reuse JavaBeans components, or forward the user to another page. Actions will be re-evaluated each time, when the page is accessed.

Slides 4 and 5

Let us understand the use of standard actions.

Use of Standard Actions 1-2

- ❖ JSP standard actions are performed when a browser requests for a JSP page.
- ❖ The properties of standard actions are as follows:
 - It use <jsp> prefix.
 - The attributes are case sensitive.
 - Values in the attributes must be enclosed in double quotes.
 - Standard actions can be either an empty or a container tag.
- ❖ Syntax:


```
<jsp:action_name
        attribute="value"
        attribute="value"/>
```

Or

```
<jsp:action_name
  attribute="Value"
  attribute="Value">
  ...
</jsp:action_name>
```

© Aptech Limited. Web Component Development Using Java/Session 9 4

Use of Standard Actions 2-2

- ❖ Various standard action tags available in JSP are as follows:

Standard Actions Tags

- <jsp:include>
- <jsp:forward>
- <jsp:param>
- <jsp:params>
- <jsp:plugin>
- <jsp:fallback>
- <jsp:text>

© Aptech Limited. Web Component Development Using Java/Session 9 5

Use slides 4 and 5 to explain the uses of standard action and different types of action elements. Action elements are basically pre-defined functions.

Tell them that the properties of standard actions in JSP are:

- It use <jsp> prefix.
- The attributes are case sensitive.
- Values in the attributes must be enclosed in double quotes.
- Standard actions can be either an empty or a container tag.

Then, explain the syntax of the action elements that conforms to the XML syntax.

Then, list the different types of action elements supported in the JSP page.

Slides 6 to 8

Let us understand <jsp:include>.

<jsp:include> 1-3

- ❖ The <jsp:include> element offers choice to include either a static or dynamic file in the current requested JSP page.
- ❖ For static file, the content is included in the calling JSP file.
- ❖ For dynamic file, it acts on a request and sends back a result that is included in the JSP page.
- ❖ When the user is not sure whether the file is static or dynamic, use <jsp:include> element, as it handles both types of files.

❖ **Syntax:**

```
<jsp:include page="weburl" flush="true" />
```

where,

- ❑ **page** specifies the relative Web address of the page to be included in the current page.
- ❑ **flush** attribute is used to flush out the data stored in the buffer. The default value is false.

© Aptech Limited. Web Component Development Using Java/Session 9

6

<jsp:include> 2-3

- ❖ The code snippet demonstrates the use of <jsp:include> action to display current date and time on the JSP page.

```
<!-- index.jsp -->
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    </head>
    <title> Dynamic Content Inclusion </title>
</html>
```

```
<!-- printdate.jsp-->
<%@page
contentType="text/html"
pageEncoding="UTF-8"
import="java.util.*"%>
<html>
    <body>
        <%
            Date today = new Date();
            out.print(today.toString());
        %>
    </body>
</html>
```

© Aptech Limited. Web Component Development Using Java/Session 9

7

©Aptech Limited

<jsp:include> 3-3

❖ Output:

© Aptech Limited. Web Component Development Using Java/Session 9 8

Use slides 6 to 8 to explain `<jsp:include>`. Tell the students that this action element is used to include the runtime response of a JSP page. It gives you choice to include either a static or dynamic file in the current requested JSP page is executed. The example of static pages is including content from the HTML page. The results are different for each type of inclusion. The `<jsp:include>` action tag works better for dynamic pages.

The `<jsp:include>` include the output generated by another file into the requested JSP. The output from the included document is inserted into the original file in place of the `<jsp:include>` tag.

The advantages of using the `<jsp:include>` are as follows:

- Automatic recompilation is guaranteed.
- Class sizes are small.
- Use of parameters.
- JSP expressions can be used in attribute values.
- Includes text from any source.
- Includes resource at request time.
- Code reusability.

Then, explain the syntax of the `<include>` element.

`<jsp:include page="weburl" flush="true" />`. The "weburl" refers to the URL of the page location that needs to be included. The value `flush` specifies that it flushes the data stored in the buffer. It can be either true or false.

The <include> element can be passed with parameter. The syntax is as follows:

```
<jsp:include page="Relative_URL_Of_Page">
  <jsp:param ... />
  <jsp:param ... />
</jsp:include>
```

Following code snippet is the example for the <include> element with parameter:

```
<html>
  ...
<body>
<h2>Passing Parameter with include</h2>
<jsp:include page="display.jsp">
  <jsp:param name="userid" value="John" />
  <jsp:param name="password" value="Rock" />
  <jsp:param name="age" value="27" />
</jsp:include>
</body>
```

Tips:

The <jsp:include> action includes the file while processing the request of the client, whereas the @include directive includes the content of the specified HTML or JSP page at the transition phase.

Slides 9 to 11

Let us understand <jsp:forward>.

<jsp:forward> 1-3

- ❖ It is used to redirect the request object containing the client request from one JSP to another target page.
- ❖ The target page can be an HTML file, another JSP file, or a Servlet.
- ❖ In the source JSP file, the code after the <jsp:forward> element is not processed.
- ❖ To pass parameter names and values to the target file, user can use a <jsp:param> clause with <jsp:forward> element.
- ❖ Syntax:

```
<jsp:forward page="url"/>
```

where,

- page specifies the relative Web address of the target page.

© Aptech Limited. Web Component Development Using Java/Session 9 9

<jsp:forward> 2-3

- ❖ The code snippet demonstrates the use of <jsp:forward> action to display current date and time on the next JSP page.

```
<!-- index.jsp -->
<%@page
contentType="text/html"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<body>
<h4><font color="BLUE">
Displaying Current Date and
Time </font></h4>
<!-- Forwards request to the
other Web resource -->
<jsp:forward
page="printdate.jsp"/>
<p><i> The request is
forwarded to the next page to
display Date and Time.
</i></p>
</body>
</html>
```

```
<!-- printdate.jsp -->
<%@page
contentType="text/html"
pageEncoding="UTF-8"
import="java.util.*"%>
<html>
<body>
<%! Date today = new
Date();%>
<b> Today is: </b>
<%
out.print(today.toString());
%>
</body>
</html>
```

© Aptech Limited. Web Component Development Using Java/Session 9 10

The screenshot shows a web page titled "<jsp:forward> 3-3". Below it is a section labeled "Output" containing a screenshot of a browser window. The browser window has the address bar showing "localhost:8084/Session_09/index.jsp". The main content area of the browser displays the text "Today is: Tue May 27 14:21:03 IST 2014". At the bottom of the slide, there is footer text: "© Aptech Limited.", "Web Component Development Using Java/Session 9", and "11".

Use slides 9 to 11 to explain `<jsp:forward>`. Tell the students that JSP forward action terminates the action of the current page and forwards a request to another resource such as a JSP page or a static page such as html. In JSP forward action, request can be forwarded with or without parameter.

The `<jsp:forward>` action is used to transfer processing from one JSP to another on the local server permanently. Mention that any content generated by the original page is discarded and the new processing begins at the second JSP.

Then, explain them the syntax of `<jsp:forward>` action tag which is as follows:

```
<jsp:forward page="URL of the another static, JSP, OR Servlet page"
/>
```

Then, explain the code snippet demonstrating the use of `<jsp:forward>` action tag. Tell them when the JSP engine executes `index.jsp` page, the request would be transferred to another JSP page named `printdate.jsp`. The `printdate.jsp` page will print the current date and time on the Web page.

Note, that both the JSP pages should be present in the same directory, otherwise, the developer has to specify the complete path of the JSP page in the `<jsp:forward>` tag.

Tips:

Whenever the processing of a page meets the `<jsp:forward>` tag, the output generated so far will be cleared. If some content has been already sent to the browser, then an `IllegalStateException` will occur. To avoid this, ensure the `<jsp:forward>` tag occurs towards the top of the page or adjust the size of the buffer using the `buffer` directive.

Following code snippet shows the use of <jsp:forward> tag with parameters:

```
<html>
<body>
    <jsp:forward page="display.jsp">
        <jsp:param name="username" value="John" />
        <jsp:param name="password" value="Rock" />
    </jsp:forward>
</body>
```

display.jsp

```
<html>
<body>
    <%
        User name is: <%=request.getParameter("name") %>      <br>
        Password is: <%=request.getParameter("password") %>  <br>
    %>
</body>
</html>
```

Slides 12 to 14

Let us understand <jsp:param>.

<jsp:param> 1-3

- ❖ It allows the user to pass one or more name and value pairs as parameters to an included or forwarded resource such as a JSP page, Servlet, or other resource that can process the parameter.
- ❖ It allows to send more than one parameters to the included or forwarded resource, user can use more than one <jsp:param> clause.
- ❖ Syntax:

```
<jsp:param name="parameterName" value="{parameterValue"
| <%= expression %>}" />
```

where,

- name attribute specifies the parameter name and that takes a case-sensitive literal string.
- value attribute specifies the parameter value and takes either a case-sensitive literal string or an expression that is evaluated at request time.

© Aptech Limited. Web Component Development Using Java/Session 9 12

<jsp:param> 2-3

- ❖ The code snippet demonstrates forwarding request to the page along with the parameter values.

```
<!-- product.jsp -->
<html>
<body>
    <jsp:include page="order.jsp" flush="true">
        <jsp:param name="currency_type" value="Dollar"/>
        <jsp:param name="amount" value="$110"/>
    </jsp:include>
</body>
</html>
```

```
<!-- order.jsp -->
<body>
<% String currency =
    request.getParameter("currency_type");
String amount = request.getParameter("amount");
%>
<b><u> Param Values </u></b> <br/><br/>
<% out.println("Currency: " + currency); %>
<br/><br/>
<% out.println("Amount: " + amount); %>
</body>
```

© Aptech Limited. Web Component Development Using Java/Session 9 13

<jsp:param> 3-3

- ❖ Output:

© Aptech Limited. Web Component Development Using Java/Session 9 14

Use slides 12 to 14 to explain <jsp:param>. Tell the students that <jsp:param> allows the developer to pass one or more name and value pairs as parameters to an included or forwarded resource using the <jsp:include> or <jsp:forward>.

Then, explain the syntax of <jsp:param> tag which is as follows:

```
<jsp: param name="param_name_here" value="value_of_parameter_here"  
/>
```

Then, explain the code snippet provided on slide 13 and 14 to show the use of <jsp:param> tag.

Slides 15 to 17

Let us understand <jsp:plugin>.

<jsp:plugin> 1-3

- ❖ It plays or displays an object, using a Java plugin that is available in the browser or downloaded from a specified URL.
- ❖ The plug-in object can be an applet or a JavaBean component that can be displayed on a JSP page.
- ❖ The <jsp:plugin> element is replaced by either an <object> or <embed> element.
- ❖ The <jsp:plugin> element provides attributes that are used for formatting Java object. These attributes are similar to HTML tag attributes.
- ❖ To pass values to the Java objects, the <jsp:param> or <jsp:params> actions can be used.

© Aptech Limited. Web Component Development Using Java/Session 9 15

<jsp:plugin> 2-3

- ❖ **Syntax:**

```
<jsp:plugin type="bean|applet" code="className"
            codebase="classFileDirectoryName" | align="alignment"
            | archive="archiveList" | height="height"
            | hspace="hspace" | jreversion="jreversion"
            name="componentName" | vspace="vspace" | width="width"
            | nspluginurl="url" | iepluginurl="url" |
            <jsp:params>
              { <jsp:param name="paramName" value= "paramValue" />
            }+
            </jsp:params>
            { <jsp:fallback> arbitrary-text </jsp:fallback> } >
</jsp:plugin>
```

where,

- ❑ Attributes specified between {} are providing configuration data for presenting the component on the Web page
- ❑ <jsp:params> element provides parameters to the Java object
- ❑ <jsp:fallback> element specifies the content to be used if the plug-in is not existing

© Aptech Limited. Web Component Development Using Java/Session 9 16

<jsp:plugin> 3-3

- The code snippet demonstrates the <jsp:plugin> element to display an applet on the JSP page.

```
<jsp:plugin type="applet" code="game.class" codebase="/html">
  <jsp:params>
    <jsp:param name="image" value="image/shape.mol" />
  </jsp:params>

  <jsp:fallback>
    <p> Unable to start the plugin. </p>
  </jsp:fallback>
</jsp:plugin>
```

Use slides 15 to 17 to explain <jsp:plugin>. Tell the students that the <jsp:plugin> displays an object that is available in the browser or downloaded from a specified URL.

If the plugin required is not present, it downloads the plugin and then executes the Java component. The Java component can be either an Applet or a JavaBean. The plugin action has many attributes that corresponds to common HTML tags used to format Java components.

Tips:

When the JSP page is translated and compiled and Java and sends back an HTML response to the client, the <jsp:plugin> element is replaced by either an <object> or <embed> element, according to the browser version. The <object> element is defined in HTML 4.0 and <embed> in HTML 3.2.

Then, explain them the syntax of the <jsp:plugin> tag mentioned on slide 16.

The following table explains the attributes of the tag:

Attribute	Description
type="bean applet"	The type of object the plug-in will execute. You must specify either bean or applet, as this attribute has no default value.
code="classFileName"	The name of the Java class file the plug-in will execute. You must include the .class extension in the name. The class file you specify should be in the directory named in the codebase attribute.
codebase="classFileDirectoryName"	Contains the Java class file the plug-in will execute. If you do not supply a value, the path of the JSP page that calls <jsp:plugin> is used.
name="instanceName"	A name for the instance of the bean or applet, which makes it possible for applets or Beans called by the same JSP page to communicate with each other.
archive="URItoArchive, ..."	A comma-separated list of pathnames that

	locate archive files that will be preloaded with a class loader located in the directory named in codebase. The archive files are loaded securely, often over a network, and typically improve the applet's performance.
	The position of the image, object, or applet. The initial height and width, in pixels, of the image the applet or bean displays, not counting any windows or dialog boxes the applet or bean brings up.
hspace="leftRightPixels" vspace="topBottomPixels"	The amount of space, in pixels, to the left and right (or top and bottom) of the image the applet or bean displays. The value must be a non-zero. Note, that hspace creates space to both the left <i>and</i> right and vspace creates space to both the top <i>and</i> bottom.
jreversion="JREVersionNumber"	The version of the Java Runtime Environment (JRE) the applet or bean requires. The default value is 1.2.
nspluginurl="URLToPlugin"	The URL where the user can download the JRE plug-in for Netscape Navigator. The value is a full URL, with a protocol name, optional port number, and domain name.
iepluginurl="URLToPlugin"	The URL where the user can download the JRE plug-in for Internet Explorer. The value is a full URL, with a protocol name, optional port number, and domain name.
<jsp:params> <jsp:param name=" <i>parameterName</i> " value=" <i>parameterValue</i> <%= expression %>}" />]+ </jsp:params>	The parameters and values that you want to pass to the applet or bean. To specify more than one parameter value, you can use more than one <jsp:param> element within the <jsp:params> element.
<jsp:fallback> <i>text message for user</i> </jsp:fallback>	A text message to display for the user if the plug-in cannot be started. If the plug-in starts but the applet or bean does not, the plug-in usually displays a popup window explaining the error to the user.

Then, explain the code snippet provided on slide 17 to display an applet on the JSP page.

Slide 18

Let us understand <jsp: fallback>.

The slide has a blue header bar with the title '**<jsp: fallback>**' in white. Below the header is a white content area. At the top of the content area, there is a bulleted list of four points. Below the list is a section titled '**Syntax:**' followed by a code snippet. At the bottom of the slide, there is some small text and a page number.

- ❖ A text message that conveys the user that a plug-in could not start is known as fallback.
- ❖ If the plug-in starts, but the applet or bean does not, the plug-in usually displays a popup window explaining the error to the user.
- ❖ The **<jsp: fallback>** action specifies an alternative message to the browser, if the browser fails to start the plug-in.
- ❖ **Syntax:**

```
<jsp: fallback> text message </jsp: fallback>
```

© Aptech Limited. Web Component Development Using Java/Session 9 18

Use slide 18 to explain <jsp: fallback>. Tell the students that <jsp: fallback> tag can be used only in the body of <jsp: plugin> tag. If <jsp: fallback> is used in any other context, it results in a compilation error.

The <jsp: fallback> tag specifies the content to be used by the client browser if the plugin cannot be started, either because the client does not support the HTML <embed> or <object> elements by the client browser or due to some other problem. If the plugin is started, but the Applet or JavaBeans component cannot be found or started, a plugin specific message will be presented. The message is a popup window reporting a `ClassNotFoundException`.

Following code snippet shows an example for <jsp: fallback> tag used within the <jsp: plugin> tag:

```
<jsp: plugin>
. . .
<jsp: fallback>
  <p>Could not load page!</p>
</jsp: fallback>
. . .
</jsp: plugin>
```

Slide 19

Let us understand the `<jsp:text>` action tag.

<jsp:text>

- ❖ It encloses contents that are displayed within the body of a JSP page or a JSP document.
- ❖ It does not have any sub-element and can appear anywhere in the JSP page where content has to be displayed in the output.
- ❖ The `<jsp:text>` element can contain expressions which are evaluated and their result is displayed in the output.
- ❖ The code snippet demonstrates the use of expression language with `<jsp:text>` element.

```

<html>
<body>
  <jsp:text> This is the product page.
  </jsp:text>
  .
  <jsp:text>Rectangle Perimeter is: ${2* 10 + 2*
20} </jsp:text>
</body>
</html>

```

© Aptech Limited. Web Component Development Using Java/Session 9 19

Use slide 19 to explain the `<jsp:text>` action tag. Tell the students that `<jsp:text>` encloses contents that are displayed within the body of a JSP page or a JSP document. The interpretation of a `<jsp:text>` element is to pass its content through to the current value of `out`.

The syntax for this action is as follows:

`<jsp:text>Template data</jsp:text>`

The body of the template contain only text and EL expressions.

Following code snippet shows an example for `<jsp:text>`:

```

<hello>
  <jsp:scriptlet>i=3;</jsp:scriptlet>
  <hi>
    <jsp:text> happy christmas santa
    </jsp:text><jsp:expression>i</jsp:expression>
  </hi>
</hello>

```

The output is:

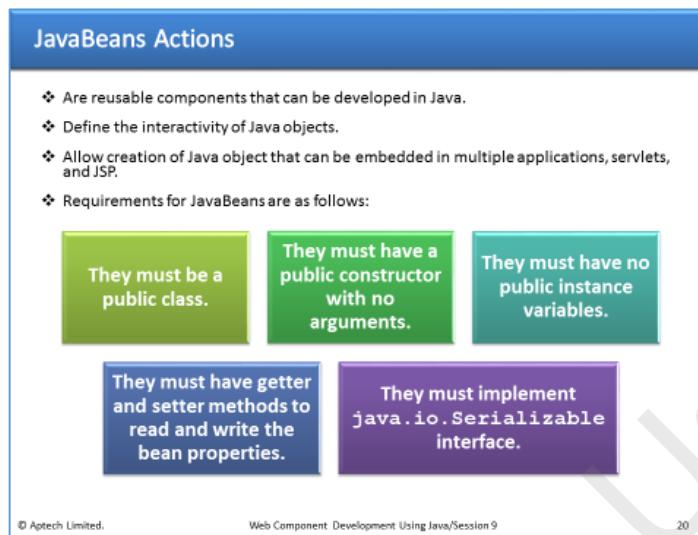
```
<hello> <hi> happy christmas santa
</hi></hello>
```

Tips:

`jsp:text` represents the XML syntax. The XML syntax allows an XML element that does not represent a standard or custom action to appear anywhere in the `jsp:text` tag.

Slide 20

Let us understand JavaBeans actions.



The slide has a blue header bar with the title "JavaBeans Actions". Below the header, there is a bulleted list of requirements for JavaBeans:

- ❖ Are reusable components that can be developed in Java.
- ❖ Define the interactivity of Java objects.
- ❖ Allow creation of Java object that can be embedded in multiple applications, servlets, and JSP.
- ❖ Requirements for JavaBeans are as follows:

The requirements are listed in four colored boxes:

- They must be a public class.
- They must have a public constructor with no arguments.
- They must have no public instance variables.
- They must have getter and setter methods to read and write the bean properties.
- They must implement `java.io.Serializable` interface.

At the bottom of the slide, there is some small text: "© Aptech Limited.", "Web Component Development Using Java/Session 9", and "20".

Use slide 20 to explain JavaBeans actions. Tell the students that JavaBean can be used in a variety of other Java based software such as applications, Servlets, or JSP pages.

JavaBeans can also have a visual component like the ActiveX controls that can be dragged-and-dropped into a Visual Basic application.

Then, explain the requirements of a JavaBean component mentioned on the slide for creating the Web applications.

Slides 21 to 23

Let us understand components of JavaBeans.

Components of JavaBeans 1-3

- ❖ JavaBeans are Java classes that include:



Properties

- It represents bean attributes that can be used to modify the appearance or behavior of the JavaBean.

Methods

- It allows implementing a bean and can be called from other components or from a scripting environment, such as JSP.

Events

- It allows communication between JavaBeans.

© Aptech Limited. Web Component Development Using Java/Session 9 21

Components of JavaBeans 2-3

- ❖ The code snippet demonstrates the JavaBean component.

```
import java.io.Serializable;
public class Person implements Serializable
{
    Private String firstName;
    // Constructor
    public Person() { }
    // Sets name
    public void setFirstName(String name)
    {
        firstName=name;
    }
    // Retrieves name
    public string getFirstName()
    {
        return firstName;
    }
    // End of Person class
```

© Aptech Limited. Web Component Development Using Java/Session 9 22

Components of JavaBeans 3-3

- ❖ The getter and setter methods are public methods defined in a JavaBean class, Person.
- ❖ These methods are also referred as accessor methods and allow retrieving and setting variable values of JavaBean properties.

Get method

It allows retrieving a variable value

Set method

It allows setting or changing the value of a property

- ❖ Syntax:

**public returnType
getPropertyName()**

where,
 returnType is the data type returned by the get method.
- ❖ Syntax:

**public void
setPropertyName(datatype
parameter)**

where,
 parameter is the property that will be set.

© Aptech Limited. Web Component Development Using Java/Session 9 23

Use slides 21 to 23 to explain the components of JavaBeans.

Tell the students that a Java class that follows certain design rules is a JavaBeans component.

A JavaBean is a Java class that:

- Should have a no-argument constructor.
- Should be Serializable.
- Should provide methods to set and get the values of the properties. These methods are also known as getter and setter methods.

Then, explain the components of JavaBean using slide 21. Based on the components of the JavaBean, using slide 22, explain them the designing of the JavaBean component.

Then, using slide 23, explain the use of Get and Set method created in the JavaBean class.

Get method – Allows retrieving a variable value.

Set method – Allows setting or changing the value of a property.

Finally, explain them how to access the Person JavaBean within a Servlet using the following code snippet:

```
 . . .
    Person p1 = new Person(); //object is created
    p1.setFirstName("Emmerson"); //setting value to the object
    System.out.println(p1.getFirstName());
```

In-Class Question:

After you finish explaining components of JavaBeans, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which method allows retrieving value from JavaBean properties?

Answer:

Getter method.

Slides 24 to 26

Let us understand declaring JavaBeans in JSP.

Declaring JavaBeans in JSP 1-3

- ❖ The `<jsp:useBean>` element is used to locate or instantiate a JavaBean component.
- ❖ Steps followed by `<jsp:useBean>` to locate or instantiate the bean are as follows:

```

graph TD
    A[Attempts to locate a bean within the scope.] --> B[Defines an object reference variable with the name.]
    B --> C[Stores a reference to it in the variable, if it retrieves the bean.]
    C --> D[Instantiates it from the specified class, if it cannot retrieve the bean.]
  
```

- ❖ Syntax:

```
<jsp:useBean id="BeanName"
              class="BeanClass"
              scope = "page|session|application|request"/>
```

where,

- id uniquely creates a name for referring the bean.
- class specifies the fully qualified class name that defines bean.
- scope specifies the scope within which the bean is available. The default scope is page if not specified.

© Aptech Limited. Web Component Development Using Java/Session 9 24

Declaring JavaBeans in JSP 2-3

Consider that the `Person` JavaBean class is created in the Web application under `com.bean` package. To access the properties of the `Person` class within the JSP page, the user need to instantiate it.

- ❖ The `jsp:useBean` action is used to include the bean in the current JSP page.
- ❖ The code snippet demonstrates the instantiation of `Person` JavaBean in the JSP page.

```
<html>
  <body>
    <jsp:useBean id="personID"
                 class="com.bean.Person" scope="request"/>
  </body>
</html>
```

- ❖ The code checks if any bean class instance with reference `personID` exists in the request scope
 - If Yes, it accesses the bean reference.
 - If No, it creates a new instance and sets it with request scope.

© Aptech Limited. Web Component Development Using Java/Session 9 25

Declaring JavaBeans in JSP 3-3

- ❖ Following is the Java equivalent code for the code snippet shown on the previous slide:

```
Person personID = (Person)
request.getAttribute("personID");
if (personID == null)
{
    personID = new Person();
    request.setAttribute("personID", personID);
}
```

© Aptech Limited. Web Component Development Using Java/Session 9 26

Use slides 24 to 26 to explain how to declare JavaBeans in JSP. Tell the students that JavaBean can be directly used in a JSP page with the help of `<jsp:useBean>` action tag.

The `<jsp:useBean>` element is used to locate or instantiate a JavaBean component. The `<jsp:useBean>` first tries to locate an instance of the bean, otherwise it instantiates the bean from a class.

Then, explain them the steps taken by the container to locate or instantiate the JavaBean that are as follows:

1. Container attempts to locate the JavaBean with the scope and name specified by the developer.
2. Defines an object reference variable with the name specified by the developer.
3. If it finds the bean, then stores a reference to it in the variable. If you specified type, gives the bean that type.
4. If it does not find the bean, instantiates it from the class you specify, storing a reference to it in the new variable. If the class name represents a serialized template, the bean is instantiated by `java.beans.Bean.instantiate`.
5. If `<jsp:useBean>` has instantiated (rather than located) the bean, and if it has body tags or elements (between `<jsp:useBean>` and `</jsp:useBean>`), executes the body tags.

Tell them that the `<jsp:useBean>` tag can be used as an empty tag or can contain body in it. The body of a `<jsp:useBean>` element often contains a `<jsp:setProperty>` element that sets property values in the bean. The body tags are only processed if `<jsp:useBean>` instantiates the bean. If the bean already exists and `<jsp:useBean>` locates it, the body tags have no effect.

Then, explain the syntax of `<jsp:useBean>` tag which is as follows:

```
<jsp:useBean id="BeanName" class="BeanClass" scope = "page|session|application|request"/>
```

Following table explains the attributes of the `<jsp:useBean>` tag:

Attribute	Description
<code>id="beanName"</code>	A variable that identifies the bean. This variable can be used in expressions or scriptlets in the JSP page.
<code>class="BeanClass"</code>	Instantiates a bean from a class, using the <code>new</code> keyword and the class constructor. The class must not be abstract and must have a public, no-argument constructor. The package and class name are case sensitive.
<code>scope="page request session application"</code>	Scope in which the bean exists and the variable named in <code>id</code> is available. The default value is <code>page</code> . <code>page</code> – Specifies that you can use the bean within the JSP page. <code>request</code> – Uses the bean from any JSP page processing the same request, until a JSP page sends a response to the client or forwards the request to another resource. <code>session</code> - Uses the bean from any JSP page in the same session as the JSP page that created the bean. <code>application</code> - Uses the bean from any JSP page in the same application as the JSP page that created the bean.

Then, explain the code snippet provided on slide 25 to demonstrate how to access Person JavaBean on the JSP page.

Tips:

You can use a <jsp:useBean> element to locate or instantiate a JavaBeans component , but not an enterprise bean. To create enterprise beans, you can write a <jsp:useBean> element that calls a bean that in turn calls the enterprise bean, or you can write a custom tag that calls an enterprise bean directly.

Slides 27 to 29

Let us understand setting value in JavaBeans.

Setting Value in JavaBeans 1-3

- ❖ The `<jsp:setProperty>` element sets the value of the properties in a bean using the bean's setter methods.
- ❖ The user must declare the JavaBean with `<jsp:useBean>` before setting a property value with `<jsp:setProperty>`.
- ❖ `<jsp:setProperty>` can be used to set property values by passing:
 - ❑ All the values accepted from the user
 - ❑ A specific value to set a specific property of the JavaBean
- ❖ Syntax:


```
<jsp:setProperty name = "BeanAlias" property =
'PropertyName" value = "Value" param =
'Parameter" />
```

where,

- ❑ name specifies the id of the bean used in `jsp:useBean` action.
- ❑ property specifies the property name of the bean.
- ❑ value specifies the explicit value to set for the property.
- ❑ param specifies the value sent in the request parameter to be assigned to a property.

© Aptech Limited. Web Component Development Using Java/Session 9 27

Setting Value in JavaBeans 2-3

- ❖ The code snippet demonstrates how to set the value for the `firstName` property of `Person` class in JSP.

```
<html>
  <body>
    <jsp:useBean id="personID"
    class="com.bean.Person" scope="request"/>
    <!-- Sets the value for the firstname
    property-->
    <jsp:setProperty name="personID"
    property="firstName"
    value="John"/>
  </body>
</html>
```

where,

- ❑ `jsp:useBean` action is used to include the `Person` class in the current JSP page.
- ❑ `jsp:setProperty` action is used to set the value, `John` to the `firstName` property for the bean instance, `personID`.

© Aptech Limited. Web Component Development Using Java/Session 9 28

Setting Value in JavaBeans 3-3

- ❖ The user can also set the value for the `firstName` by retrieving it from the request parameter sent through HTML form field. Thus, the action tag will be:
 - ❑ `<jsp:setProperty name="personID" property="firstName"/>`.
- ❖ The value of the request parameter `firstName` is retrieved and set in the bean property `firstName`.
- ❖ If the supplied request parameter name is not same as bean property name, then user can use the `param` attribute.
- ❖ For example, if the HTML form field name to accept users' first name is set as `name`, then `jsp:setProperty` action can be written as:
 - ❑ `<jsp:setProperty name="personID" property="firstName"
 param="name"/>`.

© Aptech Limited. Web Component Development Using Java/Session 9 29

Use slides 27 to 29 to explain setting value in JavaBeans. The `<jsp:setProperty>` tag sets a property value or values in a bean, using the bean's setter methods. It is important to declare the JavaBean on the JSP page, before the developer can set the value for its properties.

The value of `id` property of the `<jsp:useBean>` tag must match the value of the `name` attribute specified in the `<jsp:setProperty>`.

Then, explain the syntax of the `<jsp:setProperty>` which is as follows:

```
<jsp:setProperty name = "BeanAlias" property = "PropertyName" value
= "Value" param = "Parameter" />
```

Following table describes the attributes of the `<jsp:setProperty>` tag.

Attribute	Description
<code>name = "BeanAlias"</code>	This property specifies the name of the bean instance that has already been created or located with a <code><jsp:useBean></code> element. The value of <code>name</code> must match the value of <code>id</code> in <code><jsp:useBean></code> . The <code><jsp:useBean></code> element must appear before <code><jsp:setProperty></code> in the JSP page.
<code>property = "PropertyName"</code>	Stores the value of the request parameter in the bean property. The bean property is defined as a variable with a matching setter method in the Java Bean class. Note that the name of the bean property must match the name of the request parameter.
<code>value = "Value"</code>	Sets the new value for the bean property in the JSP page. The value can be a String or an expression that is evaluated at the runtime.
<code>param = "Parameter"</code>	Specifies the name of the request parameter whose value needs to be set in the bean property.

Then, explain the code snippet mentioned on slide 28 to set the property of the Person JavaBean.

Tips:

- Normally, the values sent by the client in the request parameter are of type string. However, the String values are converted into the appropriate data type when stored in the bean property.
- You can use the `property="*"`. This means, the values of all the request parameters are set in bean properties. However, the order in which the values are appearing in the HTML form must with the properties set in the JavaBean class.

Slides 30 and 31

Let us understand retrieving value from JavaBeans.

Retrieving Value from JavaBeans 1-2

- ❖ The `<jsp:getProperty>` element retrieves a bean property value using the getter methods and displays the output in a JSP page.
- ❖ Before using `<jsp:getProperty>` element, the user must create or locate a bean with `<jsp:useBean>`.
- ❖ Syntax:

```
<jsp:getProperty name="BeanAlias"
    property="PropertyName" />
```

where,

- ❑ name specifies the id of the bean specified in the `jsp:useBean` action.
- ❑ property specifies the property name from which the value is to be retrieved.

© Aptech Limited. Web Component Development Using Java/Session 9 30

Retrieving Value from JavaBeans 2-2

- ❖ The code snippet demonstrates how to retrieve value of the `firstName` property of `Person` class in JSP.

```
<html>
  ...
<body>
  <jsp:useBean id="personID" class="com.bean.Person" scope="request"/>
  Name is: <jsp:getProperty name="personID"
    property="firstName"/>
</body>
</html>
```

- ❖ In the code:
 - ❑ `useBean` is used to access the `Person` class instance in the current JSP page.
 - ❑ `jsp:getProperty` action is used to retrieve the value of the `name` property from the JavaBean instance.
- ❖ The equivalent Java code for the `jsp:getProperty` is as follows:


```
out.print(personID.getFirstName());
```

© Aptech Limited. Web Component Development Using Java/Session 9 31

Use slides 30 and 31 to explain how to retrieve value from JavaBeans. Tell the students that apart from `jsp:getProperty`, there are several methods to retrieve JavaBeans component properties. One of them is using expression. Both `jsp:getProperty` element and an expression converts the value of the property into a String and insert the value into the current implicit `out` object. You must create or locate a bean with `<jsp:useBean>` before you use the `<jsp:getProperty>` tag on the JSP page.

The syntax for the expression for retrieving Value from JavaBeans is as follows:

```
<%= beanName.getPropertyName() %>
```

For both `jsp:getProperty` element and an expression, the value of the `name` property in the `<jsp:getProperty>` tag must be the same as that specified for the `id` attribute in a `useBean` element, and there must be a getter method in the JavaBean class.

To retrieve the value of a property without converting it and inserting it into the `out` object, scriptlet should be used. The syntax for using scriptlet is as follows:

```
<% Object a = beanName.getPropertyName(); %>
```

Then, explain the code snippet present on slide 31 to use the `<jsp:getProperty>` tag.

Tips:

If you use the `<jsp:getProperty>` tag to retrieve the property whose value is set to `NULL`, then a `NullPointerException` is thrown.

Following code snippet shows how to access a JavaBeans from a JSP with its properties.

```
<HTML>
<HEAD>
<TITLE> Example: Simple Java Bean</TITLE>
<jsp:useBean id="studentBean" scope="page"
class="com.acme.mybean.StudentBean" /> </HEAD>
<BODY>
<%
String last = "Roy";
%>
<%-- Set bean properties --%>
<jsp:setProperty name="studentBean" property="firstName"
value="Rivaldo" />

<jsp:setProperty name="studentBean" property="lastName"
value="<%=last%>" />

<%-- Get bean properties --%>
<P>
<jsp:getProperty name="studentBean" property="fullName" /> </P>
</BODY>
</HTML>
```

Slide 32

Let us understand accessing JavaBeans within scriptlets.

Accessing JavaBeans within Scriptlets

- ❖ A user can access JavaBeans from scripting element in different ways.
- ❖ The `jsp:getProperty` and expression convert the value into a string and insert it into an implicit `out` object.
- ❖ To retrieve the value of a property without converting it and insert it into the `out` object, user must have to use a scriptlet.
- ❖ Scriptlets are very useful for dynamic processing.
- ❖ Using custom tags is a better approach to access object properties and perform flow control.
- ❖ The code snippet demonstrates how to create the JavaBean instance and access its properties in JSP.

```
<html>
  <body>
    // Instantiating Book bean of package pkg using scriptlet
    <% pkg.Book book1 = new pkg.Book(); %>
    // Retrieves the title property using bean getter method
    <%= book1.getTitle() %>
    // Set title property to a new value
    <% book1.setTitle("Guide to Servlets and JSP"); %>
  </body>
</html>
```

© Aptech Limited. Web Component Development Using Java/Session 9 32

Use slide 32 to explain how to access JavaBeans within scriptlets. Tell the students that the scriptlets contains a code fragment valid in the page scripting language. When JSP processes the request, scriptlets are executed at request time. Although, scriptlets are very useful for dynamic processing. However, using JSP action tags to access object properties and perform flow control is considered to be a better approach.

Following code snippet shows accessing a JavaBean within the scriptlets on the JSP page:

```
<%@ page import="com.java2s.Employee" %>
<%
  Employee myEmployee = (Employee)
  session.getAttribute("myEmployeeBean");
  if ( myEmployee == null)
  {
    myEmployee = new Employee();
    myEmployee.setName("Joel");
    session.setAttribute("myEmployeeBean", myEmployee);
  } // end of if ()
%>

<html>
  <head><title>JavaBean usage with scriptlets (1)</title></head>
  <body>
    This page creates a JavaBean if you don't already have one.<P></P>
    Click <a href="page2_scriptlet.jsp">here</a> to go to a page that
    retrieves it.
  </body>
</html>

//page2_scriptlet.jsp
<%@ page import="com.java2s.Employee" %>
<%
```

```

Employee myEmployee = (Employee)
session.getAttribute("myEmployeeBean");
%>

<html>
<head><title>JavaBean usage with scriptlets (2) </title></head>
<body>
This page retrieves a JavaBean, and its properties.<P>
<table border="1">
<th>JavaBean property</th><th>Value</th>
<tr><td>id</td> <td><%= myEmployeeBean.getId()%></td></tr>
<tr><td>designation</td> <td><%= myEmployeeBean.getDesignation()%></td></tr>
<tr><td>name</td><td><%= myEmployeeBean.getName()%></td></tr>
<tr><td>department</td> <td><%= myEmployeeBean.getDepartment()%></td></tr>
</table>
</body>
</html>

// JavaBean usage - useBean and setProperty tags
<jsp:useBean id="myEmployeeBean"
    class="com.java2s.Employee"
    scope="session">
<jsp:setProperty name="myEmployeeBean" property="id" value="42"/>
<jsp:setProperty name="myEmployeeBean" property="name" value="Ruth"/>
<jsp:setProperty name="myEmployeeBean" property="designation" value="accountant" />
<jsp:setProperty name="myEmployeeBean" property="department" value="finance" />
</jsp:useBean>

<html>
<head><title>JavaBean usage - useBean and setProperty tags</title></head>
<body>
This page creates a JavaBean if you don't already have one.<P></P>
Click <a href="useAndSet2.jsp">here</a> to go to a page that retrieves it.
</body>
</html>

// useAndSet2.jsp
<html>
<head><title>JavaBean usage - getProperty tag</title></head>
<body>

```

This page retrieves a JavaBean, and its properties.<P>

```
<table border="1">
    <th>JavaBean property</th><th>Value</th>
    <tr><td>id</td>    <td><jsp:getProperty name="myEmployeeBean"
property="id" /></td></tr>
    <tr><td>designation</td> <td><jsp:getProperty
name="myEmployeeBean" property="designation" /></td></tr>
    <tr><td>name</td><td><jsp:getProperty name="myEmployeeBean"
property="name" /></td></tr>
    <tr><td>department</td> <td><jsp:getProperty
name="myEmployeeBean" property="department" /></td></tr>
</table>
</body>
</html>
```

Slide 33

Let us understand accessing non-string data type properties.

Accessing Non-string Data Type Properties

- ❖ The `jsp:setProperty` action is used to set the properties of a bean by the values of the request parameter.
- ❖ The JSP container converts the string values into non-string values by the attribute values that evaluate the correct data type to set the property value.
- ❖ Some of the conversions from string to appropriate data type is done by using Java wrapper classes. For example:

```
String ageStr = request.getParameter("age");
int ageInt = Integer.valueOf(ageStr);

String amtStr = request.getParameter("Amount");
double amtDouble = Double.valueOf(amtStr);
```

© Aptech Limited. Web Component Development Using Java/Session 9 33

Use slide 33 to explain accessing non-string data type properties. The JSP container converts the string values into non-string values by the attribute values that evaluate the correct data type to set the property value.

Slide 34

Let us understand how to access indexed properties.

Accessing Indexed Properties

- ❖ An indexed property is an array of properties or objects that supports a range of values.
- ❖ It enables the accessor to specify an element of a property to read or write.
- ❖ If there is an indexed property 'Tomy' of type string, it may be possible from a scripting environment to access an individual indexed value using the index.
- ❖ For example, `b.Tomy[3]` and also to access the same property as an array using `b.Tomy`.
- ❖ The indexed getter and setter methods throw a runtime exception `java.lang.ArrayIndexOutOfBoundsException`, if an index is used that is outside the current array bounds.
- ❖ The value assigned to an indexed property must be an array.
- ❖ The code snippet shows how to declare the getter and setter method for the indexed property `studentRegister`.

```

// Retrieves indexed properties
public StudentRegister[] getStudentRegister()

// Sets indexed properties
public void setStudentRegister(StudentRegister int[])

```

© Aptech Limited. Web Component Development Using Java/Session 9 54

Use slide 34 to explain how to access indexed properties. Tell them that an indexed property is an array of properties or objects that supports a range of values. This property enables the accessor to specify an element of a property to read or write.

If there is an indexed property 'Tomy' of type string, it may be possible from a scripting environment to access an individual indexed value using the index. For example, `b.Tomy[3]` and also to access the same property as an array using `b.Tomy`.

Tips:

The developer can use `<jsp:setProperty>` to set the value of an indexed property in a bean. The indexed property must be an array of any of the data types. However, the developer cannot use `<jsp:getProperty>` to retrieve the values of an indexed property on the JSP page.

Slide 35

Let us summarize the session.

Summary

- ❖ JSP standard actions are XML like tags that are processed when a browser requests for a JSP page.
- ❖ Standard actions in the JSP standard library use the <jsp> prefix. Various standard actions are available that include <jsp:include>, <jsp:forward>, <jsp:param>, <jsp:params>, <jsp:plugin>, <jsp:fallback>, and <jsp:text>.
- ❖ JavaBeans are reusable components that can be developed in Java. These are platform independent and define the interactivity of Java objects in the applications.
- ❖ JavaBeans component controls access to the private properties of a class by providing public methods.
- ❖ The jsp:useBean action is used to create a reference and include an existing JavaBean component in JSP.
- ❖ The jsp:getProperty and jsp:setProperty actions act as getter and setter methods to access and retrieve values from the JavaBean components.
- ❖ An indexed property is an array of properties or objects that supports a range of values. This property enables the accessor to specify an element of a property to read or write.

© Aptech Limited. Web Component Development Using Java/Session 9 55

In slide 35, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

9.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Model-View-Controller architecture explained in the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 10 – Model-View-Controller Architecture

10.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

10.1.1 Objectives

By the end of this session, the learners will be able to:

- Describe the use of JSP models in Web applications
- Explain JSP Model 1
- Explain JSP Model 2
- Explain the Model-View-Controller architecture
- Explain the relationship between the components of MVC
- Explain Controller and its purpose in MVC
- Explain View and its purpose in MVC
- Explain Model and its purpose in MVC
- Develop a Web application based on MVC architecture

10.1.2 Teaching Skills

To teach this session successfully, you should be aware of JSP Model 1 and Model 2 architectures. You should have in-depth knowledge about the Model-View-Controller (MVC) architecture and the relationship between the components of MVC. Also, familiarize yourself with the purpose of various components in MVC architecture. You should also develop a Web application based on MVC architecture.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives
<ul style="list-style-type: none">❖ Describe the use of JSP models in Web applications❖ Explain JSP Model 1❖ Explain JSP Model 2❖ Explain the Model-View-Controller architecture❖ Explain the relationship between the components of MVC❖ Explain Controller and its purpose in MVC❖ Explain View and its purpose in MVC❖ Explain Model and its purpose in MVC❖ Develop a Web application based on MVC architecture

© Aptech Ltd.

Web Component Development Using Java/Session 10

2

Tell them that they will be introduced to JSP Model 1 and JSP Model 2 architecture. A detailed explanation of the Model-View-Controller (MVC) architecture will be given. The relationship between the Model, View, and Controller components and their purpose in MVC architecture will be described in detail. Finally, the practical implementation of MVC pattern in a Web application is discussed.

10.2 In-Class Explanations

Slide 3

Let us understand about JSP specification.

Introduction

- ❖ Sun Microsystems has provided the JSP specification to address the problem of tightly-coupled presentation and business logic in Servlets.
- ❖ Based on the popularity and benefits of JSP in Web development, the commonly used approaches using JSP were identified.
- ❖ These approaches are also referred to as JSP models.

The diagram illustrates the evolution of web development. It starts with a circle labeled "JAVA" on a teal base, followed by a horizontal arrow pointing to another circle labeled "JSP Model 1" on a blue base, which then points to a third circle labeled "JSP Model 2" on a darker blue base.

© Aptech Ltd. Web Component Development Using Java/Session 10 3

Use slide 3 to give background information about the development of JSP specification. Tell the students that in most of the Web applications, the user interaction is through forms which are used as an interface for storing and retrieving data from a data store. With time, the approaches to develop such Web applications have changed. For example, in order to improve the application performance and to reduce the Line of Code (LOC), the programmers tend to integrate code for designing and data access in the Web application. This helps to achieve simplicity and flexibility.

However, there are significant problems in implementing it. One such problem might be the more frequent change in user interface as compared to the persistence logics leading to the recompilation of whole application.

As a solution, Sun Microsystems provided the JSP specification which addresses the problem of tightly-coupled presentation and business logic in Servlets. Based on the popularity and benefits of JSP in Web development, the commonly used approaches using JSP were identified. These approaches are also referred to as JSP models.

Tips:

Problem in Servlet technology, Servlet needs to recompile if any designing code is modified. It doesn't provide separation of concern. Presentation and Business logic are mixed up.

Slide 4

Let us understand JSP Models.

JSP Models

- ❖ There are two types of programming models for developing Web application.

JSP Model 1

- The Model 1 architecture is very simple.
- The HTML or JSP page sends request along with the data to Web container.
- The Web container invokes the mapped Servlet which handles all responsibilities for the request.
- The responsibilities include processing the request, validating data, handling the business logic, and generating a response back to browser.

JSP Model 2 (Model-View-Controller)

- It provides a clear separation of application responsibilities.
- A central servlet, known as the Controller, receives all requests for the application from JSP.
- The Controller works with the Model to prepare any data needed by the View and forwards the data back to the JSP.
- The business and presentation logic are separated from each other, which help to reuse the logic.

© Aptech Ltd. Web Component Development Using Java/Session 10 4

Use slide 4 to introduce JSP models. Tell them that the JSP specification presents two approaches for developing Web applications using JSP technology. These approaches are termed as Model 1 and Model 2 architectures.

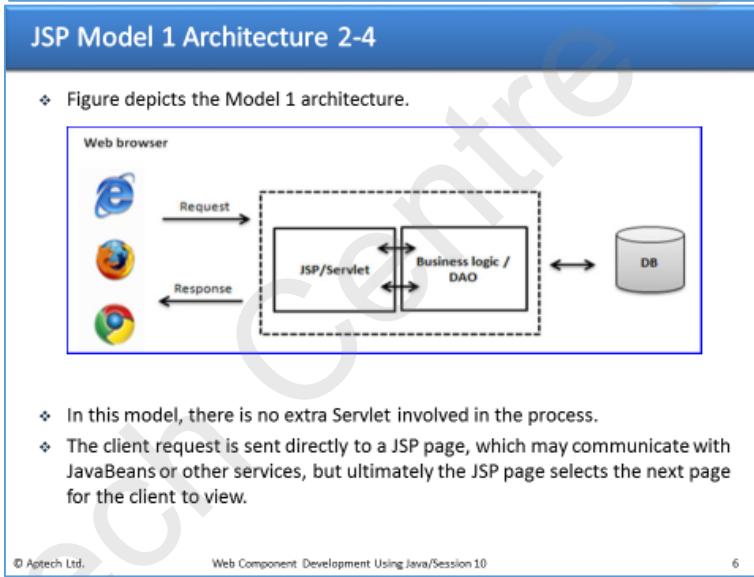
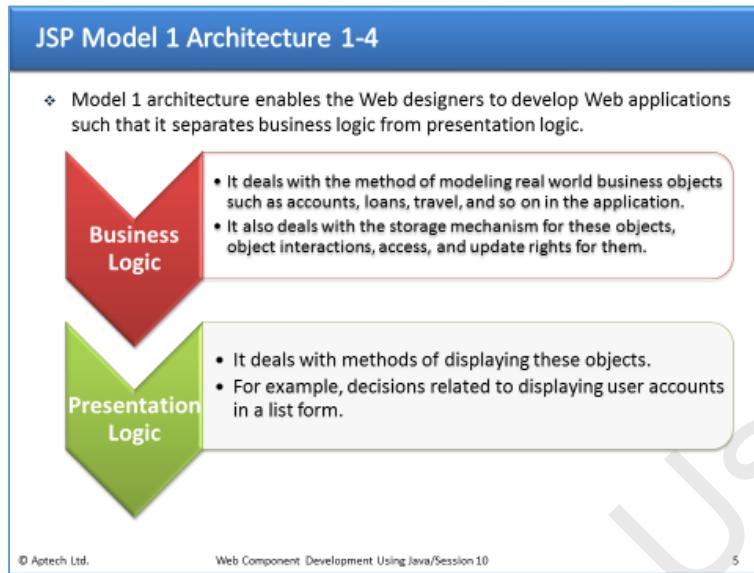
In the Model 1 architecture, the design of the Web application is based on the JSP page. Thus, JSP page acts as a focal point for the entire application.

Termed Model 1 architecture, the JSP page not only contains the display elements to output HTML, but is also responsible for extracting HTTP request parameters, call the business logic (implemented in JavaBeans, if not directly in the JSP), and handle the HTTP session. Although Model 1 is suitable for simple applications, this architecture usually leads to a significant amount of scriptlets (Java code embedded within HTML code in the JSP page), especially if there is a significant amount of request processing to be performed.

Then, explain them that Model 2 or commonly known as Model-View-Controller (MVC) paradigm applied to web applications lets you separate display code from flow control logic. Thus, this solves many problem of Model 1, by providing a clear separation of application responsibilities.

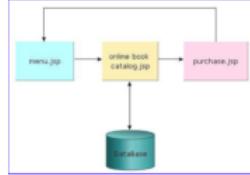
Slides 5 to 8

Let us understand JSP Model 1 architecture.



JSP Model 1 Architecture 3-4

- ❖ JSP Model 1 architecture has a page-centric architecture.
- ❖ Page-centric architecture:
 - The application is composed of a series of interrelated JSP pages and these JSP pages handle all aspects of application.
 - The business process logic and control decisions are hard-coded inside JSP pages in the form of JavaBeans, scriptlets, and expressions.
- ❖ Figure shows an example of JSP Model 1 architecture for an online shopping Web application with all JSP pages.



© Aptech Ltd.

Web Component Development Using Java/Session 10

7

JSP Model 1 Architecture 4-4

- ❖ **Advantages of JSP Model 1:**
 - It makes development easier as there are no Servlets involved in the application.
- ❖ **Disadvantages of JSP Model 1:**
 - As business logic and presentation logic are tied together, it is difficult for building and maintaining a complex enterprise application.
 - Each of the JSP pages is individually responsible for control logic, application logic, and also to present results to the user. This makes the JSP Model 1 more dependent and less extensible.

© Aptech Ltd.

Web Component Development Using Java/Session 10

8

Use slides 5 to 8 to explain JSP Model 1 architecture. The purpose of Model 1 architecture is to enable Web designers to develop Web applications such that it separates business logic from presentation logic.

- Business logic deals with the method of modeling real world business objects such as accounts, loans, travel, and such others in the application. It also deals with the storage mechanism for these objects, object interactions, access, and update rights for them.
- Presentation logic deals with methods of displaying these objects. For example, decisions related to displaying user accounts in a list form.

Use the figure shown on slide 6 to explain the Model 1 architecture. Tell them JSP Model 1 architecture has a page centric architecture. In this architecture, the application is composed of a series of interrelated JSP pages and these JSP pages handle all aspects of application including presentation, control, and the business logic.

In page-centric architecture, the business process logic and control decisions are hard-coded inside JSP pages in the form of JavaBeans, scriptlets, and expressions.

Use the figure shown on slide 7 to explain an example of JSP Model 1 architecture for an online shopping Web application with all JSP pages. Tell them that, in the example, the JSP page alone is responsible for processing the incoming request and replying back to the client.

Use slide 8 to describe the advantages and disadvantages of JSP Model 1. Tell them that navigation control in this model is distributed as each and every page encompasses the logic to define the next page. If any change is made to the page name that defines the other pages, then changes will have to be made to all the other pages also. This will lead to maintenance issues. In addition, a developer will have to spend more time and effort to develop custom tags in JSP. This model is ideal for small applications only, as it is less extensible.

Slides 9 to 11

Let us understand JSP Model 2 Architecture.

JSP Model 2 Architecture 1-3

- ❖ Model 2 architecture is an approach used for developing a Web application.
- ❖ It separates the Business logic from the Presentation logic.
- ❖ The Model 2 has an additional component - a Controller.
- ❖ Figure shows the Model 2 architecture.

```
graph LR; WB[Web Browser] -- Request --> SC[Servlet (Controller)]; SC --> JHV[JSP/HTML (View)]; SC <--> BL[Business logic / DAO (Model)]; BL <--> DB[DB]; JHV -- Response --> WB;
```

© Aptech Ltd. Web Component Development Using Java/Session 10 9

JSP Model 2 Architecture 2-3

- ❖ In Model 2 architecture, a Servlet acts as a Controller and therefore, it has a Servlet-centric architecture.
- ❖ **Servlet - Controller:**
 - Is responsible to process the incoming request and instantiate a Model - a Java object or a bean to compute the business logic.
 - Is responsible for deciding to which JSP page the request should be forwarded.
- ❖ **JSP - View:**
 - Is responsible for handling the View component.
 - Retrieves the objects created by the Servlet.
 - Extracts dynamic content for insertion within a template for display.

© Aptech Ltd. Web Component Development Using Java/Session 10 10

JSP Model 2 Architecture 3-3

- ❖ **Advantages of Model 2:**
 - Web applications based on this model are easier to maintain and extendable.
 - Testing is easy in model 2.

© Aptech Ltd. Web Component Development Using Java/Session 10 11

Use slides 9 to 11 to explain JSP Model 2 architecture.

Use the figure shown slide 9 to explain the Model 2 architecture. Model 2 has an additional component – a Controller. JSP page is responsible for handling the View component, it retrieves the objects created by the Servlet, and extracts dynamic content for insertion within a template for display.

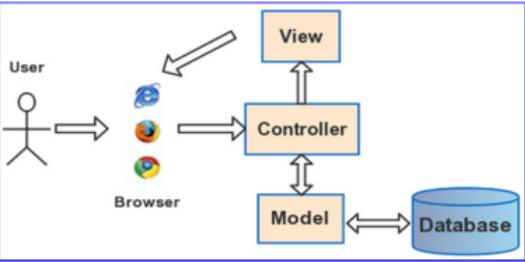
Use slide 11 to explain the advantages and disadvantages of Model 2 architecture. Tell them that Web applications based on this model are easier to maintain and extendable, as business and presentation logic are separated from each other. In this mode, navigation control is centralized and only the controller contains the logic to determine the next page. So this is easier to maintain. The disadvantage of this model is that if any change is made to the Controller code, then the class has to be recompiled and the application needs to be redeployed.

Slides 12 and 13

Let us understand Model-View-Controller.

Model-View-Controller (MVC) 1-2

- ❖ MVC is a software architectural pattern.
- ❖ This pattern divides the application logic from User Interface.
- ❖ The division permits independent development, testing, and maintenance of each component.
- ❖ Figure shows a brief overview of the components under the MVC architecture.



© Aptech Ltd. Web Component Development Using Java/Session 10 12

Model-View-Controller (MVC) 2-2

- ❖ MVC model divides the Web based application into three layers:

Controller

- Manages the flow of data between the Model layer and the View layer.
- Calls methods in the Model to fulfil the requested action.
- After the action has been taken on the data in Model, the Controller is responsible for redirecting the appropriate view to the user.

View

- Is used to generate the response to the browser, what the user sees.
- Are simple JSP or HTML pages.

Model

- Is a layer between the Controller and the database.
- Contains the business logics and functions that manipulate the business data.
- Can access the functionalities encapsulated in the Model.

© Aptech Ltd. Web Component Development Using Java/Session 10 13

Use slides 12 and 13 to Model-View-Controller (MVC) architectural pattern.

Use the figure shown on slide 12 to explain the components under the MVC architecture.

Use slide 13 to describe the components of MVC. Begin by telling them that at the core of the MVC architecture are the Controller components. The Controller is typically a servlet that receives requests from the browser (or any other source) and manages the flow of data between the Model layer and the View layer. The Controller processes the user requests. It processes the user request and based on the action, the Controller calls methods in the Model to fulfil the requested action and after the action has been taken on the data in model, the Controller is responsible for redirecting the appropriate view to the user.

In addition, tell them that the MVC pattern has been around since 1979. Trygve Reenskaug, an engineer who was working on a language called Smalltalk at XEROX, first described it. He named this concept as 'Thing Model View Editor'. The MVC model can be found in UI toolkits such as Nokia's Qt, Apple's Cocoa, Java Swing, and MFC library.

MVC can be broken down into three elements:

- **Model** - The model represents data and the rules that govern access to and updates of this data. In enterprise software, a model often serves as a software approximation of a real-world process.
- **View** - The view renders the contents of a model. It specifies exactly how the model data should be presented. If the model data changes, the view must update its presentation as needed. This can be achieved by using a *push model*, in which the view registers itself with the model for change notifications, or a *pull model*, in which the view is responsible for calling the model when it needs to retrieve the most current data.
- **Controller** - The controller translates the user's interactions with the view into actions that the model will perform. In a stand-alone GUI client, user interactions could be button clicks or menu selections, whereas in an enterprise Web application, they appear as GET and POST HTTP requests. Depending on the context, a controller may also select a new view -- for example, a Web page of results -- to present back to the user.

In-Class Question:

After you finish explaining MVC architecture, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



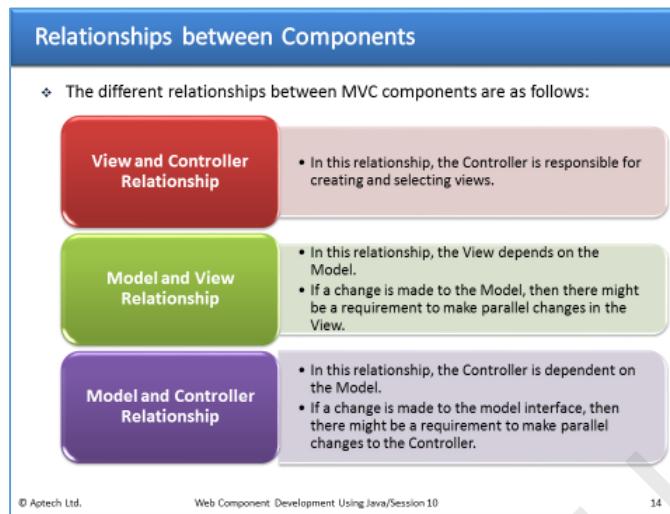
Which component in the MVC architecture receives all requests for the application from JSP?

Answer:

Controller

Slide 14

Let us understand relationships between components.



Use slide 14 to explain the relationship between MVC components.

View and Controller Relationship

In this relationship, the Controller is responsible for creating and selecting views. The relationship between View and Controller is one-to-one. The View contains all the user interface elements through which the user interacts.

Model and View Relationship

In this relationship, the View depends on the model. If a change is made to the model then there might be a requirement to make parallel changes in the View. The View interacts with the model in two ways: It listens for update messages, and it reads from the model. The View never writes to the model. Every view keeps a reference to its model. Because a view knows about its model, but a model doesn't know about a view, a single model can act as the model for many views.

Model and Controller Relationship

In this relationship, the Controller is dependent on the model. If a change is made to the model interface then there might be a requirement to make parallel changes to the Controller. The controller is responsible for updating the model when necessary, based on user input or system events.

Slide 15

Let us understand MVC in Web applications.

MVC in Web Applications

- ❖ While developing a Web application using MVC architecture, the different components and their roles are as follows:

Model encapsulates data and business logic using JavaBean components or Plain Old Java Object (POJO), a database API, or an XML file.

View shows the current state of the Model using an HTML or a JSP page.

Controller updates the state of the Model and generates one or more views using Servlet.

© Aptech Ltd. Web Component Development Using Java/Session 10 15

Use slide 15 to explain the role of the MVC components in Web applications.

Explain each component's role in Web applications with an example. Consider a scenario, when you login to a shopping Website, the Controller will communicate to the Web application to load the login form View. When you enter the login ID and password, the Controller will load the model that handles logins, which will check if the username and password match with what is saved in the system. If it matches, the Controller will let your request move on to the first page of the shopping Website.

The View is the visible interface that the user interacts with, like the input forms, displayed information, buttons, and so on. The Controller requests the View after interacting with the model, which then collects the information that is to be displayed in the particular view.

In the shopping Website example, the Model will collect the username and password given to it by the controller, verifies the data with the saved information in the database, and then renders the appropriate view. In case, a wrong user ID or password is entered, the Model will communicate to the Controller that the data entered was incorrect, and the Controller will communicate to the view to display an error message, like **You have not entered a valid user ID or password.**

Model encapsulates data and business logic using JavaBean components or Plain Old Java Object (POJO), a database API, or an XML file. View shows the current state of the Model using an HTML or a JSP page. Controller updates the state of the Model and generates one or more views using Servlet.

Slides 16 to 23

Let us understand implementation of MVC pattern.

Implementation of MVC Pattern 1-8

- ❖ **Scenario:** In a Web application, you have to develop login service which will validate login details and accordingly display the appropriate JSP page.
- ❖ To design such service, we will use the MVC pattern.
- ❖ The code snippet develops the JSP pages required to handle the presentation and result pages.

```
<!-- login.jsp-->
<html>
  <head>
    <title>MVC Example</title>
  </head>

  <body>
    <form action="LoginController" method="post">
      Enter username : <input type="text" name="username"> <BR>
      Enter password : <input type="password" name="password"> <BR>
      <input type="submit" />
    </form>
  </body>
</html>
```

© Aptech Ltd.

Web Component Development Using Java/Session 10

16

Implementation of MVC Pattern 2-8

```
<!-- success.jsp -->
<html>
  <head>
    <title>Success</title>
  </head>
  <body>
    Welcome, you have successfully login.
  </body>
</html>
```

```
<!-- error.jsp -->
<html>
  <head>
    <title>Error</title>
  </head>
  <body>
    Login failed, please try again.
  </body>
</html>
```

© Aptech Ltd.

Web Component Development Using Java/Session 10

17

Implementation of MVC Pattern 3-8

- ❖ Next, we will design a Controller that takes request from the user, this is the Servlet class.
- ❖ The servlet class calls the Model which is JavaBean.
- ❖ The code snippet demonstrates the design of the Controller named LoginController.java.

```
package com.mvc.Controller;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.mvc.model.LoginModel;

public class LoginController extends HttpServlet {
  RequestDispatcher rd = null
  protected void doPost(HttpServletRequest request,
  HttpServletResponse response) throws ServletException,
  IOException {
    // Receive the values from the request parameters
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    // Instantiate the LoginModel JavaBean
    LoginModel login = new LoginModel();
```

© Aptech Ltd.

Web Component Development Using Java/Session 10

18

Implementation of MVC Pattern 4-8

```
// Verify the login credentials from model
String result = login.authenticate(username, password);
/**
 * Dispatch the control based on the value of the
 * result variable */
if (result.equals("success")) {
    rd = request.getRequestDispatcher("/success.jsp");
} else {
    rd = request.getRequestDispatcher("/error.jsp");
}
// Forward the response to appropriate JSP
rd.forward(request, response);
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 10

19

Implementation of MVC Pattern 5-8

- The code snippet demonstrates the design of LoginModel class.

```
package com.mvc.model;
public class LoginModel {
    public String authenticate(String username, String
password) {

        // validate the login credentials with the values
        if ("username".equalsIgnoreCase(username))
            && ("password".equals(password)) {
                return "success";
            } else {
                return "failure";
            }
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 10

20

Implementation of MVC Pattern 6-8

- The code snippet shows the deployment descriptor, web.xml used to configure the servlet in the Login application.

```
<!-- web.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<!-- web.xml -->
<web-app>
    <servlet>
        <servlet-name>LoginController</servlet-name>
        <servlet-class>com.mvc.Controller.LoginController</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>LoginController</servlet-name>
        <url-pattern>/LoginController</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>login.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

© Aptech Ltd.

Web Component Development Using Java/Session 10

21

Implementation of MVC Pattern 7-8

- Figure shows the request to the URL `localhost:8080/TestServlet`.

This displays the `login.jsp` with the username and password fields.

© Aptech Ltd. Web Component Development Using Java/Session 10 22

Implementation of MVC Pattern 8-8

- Figure shows the `success.jsp`.

© Aptech Ltd. Web Component Development Using Java/Session 10 23

Use slides 16 to 23 to explain the implementation of MVC pattern with the help of a scenario of developing a login service for a Web application.

Use the Code Snippets in slides 16 and 17 to show and explain the development of the JSP pages required to handle the presentation and result pages. Tell them that the code shows the designing of three JSP pages namely, `login.jsp`, `success.jsp`, and `error.jsp`. The login page displays the form to the user for filling the user name and password. The success page will be displayed when the validation of the details is successful. Otherwise, the error page is displayed to the user.

Use the Code Snippet in slides 18 and 19 to demonstrate and explain the design of the Controller named `LoginController.java`. Tell them that in the code, the user details sent from the `login.jsp` page are received through `request.getParameter()` method. Then the values are sent to the JavaBean named `LoginModel` which will validate the values and return the appropriate string. Finally, based on the obtained string, the request to appropriate JSP page is sent.

Use the Code Snippet in slide 20 to demonstrate and explain the design of `LoginModel` class.

Use the Code Snippet in slide 21 to show and explain deployment descriptor, `web.xml` used to configure the servlet in the Login application. Tell them that the `web.xml` is placed in WEB-INF

folder and describes the deployment of an application to run servlet and JSP pages. The `login.jsp` page is the default page to open on sending request to the Web application.

Use the figure shown on slide 22 to display the request to the URL, `localhost:8080/TestServlet`. Tell them that this displays the `login.jsp` with the username and password fields. In addition, tell them that, as shown in the figure, the username and password values are sent in request to the Controller, that is, `LoginController Servlet` which invokes the JavaBean method for validating user entered details. As the entered details are correct, the Controller dispatches the request to `success.jsp` page to display the response in the browser.

Use the figure shown on slide 23 to show displaying the `success.jsp`.

Slide 24

Let us summarize the session.

Summary

- ❖ The JSP specification presents two approaches for developing Web applications namely, JSP Model I and JSP Model II.
- ❖ JSP Model II is also known as MVC.
- ❖ MVC is a software design pattern, which can be used to design medium and large sized applications.
- ❖ MVC has three components as follows:
 - ❑ Model
 - ❑ View
 - ❑ Controller
- ❖ In MVC Web application, servlet acts as Controller, which receives the request from client.
- ❖ The View handles presentation of the content on the Web page and could be an HTML file or a JSP file.
- ❖ The Model component contains the business logics and functions that manipulate the business data.

© Aptech Ltd. Web Component Development Using Java/Session 10 24

In slide 24, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

10.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the JSP expression language, accessing various operators, functions, and objects using expression language. Also, concept of boxing and unboxing, coercing a value to string or number type that are offered with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 11 – JSP Expression Language

11.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

11.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain how to use script expressions in JSP
- Describe the implicit objects used in EL
- Describe the various operators used in EL
- Explain how to create static method and tag library descriptor using EL
- Explain how to modify deployment descriptor using EL
- Explain how to access EL functions within JSP
- Explain the concept of boxing and unboxing
- Explain how to coerce a value to string or number type

11.1.2 Teaching Skills

To teach this session successfully, you should be aware with the JSP Expression Language (EL). You should aware yourself with the working of EL to access various implicit objects, operators, and functions on the JSP page. You should know how to create static method and tag library descriptor and how to modify deployment descriptor using EL. In addition, aware yourself with the concept of boxing and unboxing and also how to coerce a value to string or number type.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❖ Explain how to use script expressions in JSP
- ❖ Describe the implicit objects used in EL
- ❖ Describe the various operators used in EL
- ❖ Explain how to create static method and tag library descriptor using EL
- ❖ Explain how to modify deployment descriptor using EL
- ❖ Explain how to access EL functions within JSP
- ❖ Explain the concept of boxing and unboxing
- ❖ Explain how to coerce a value to string or number type

© Aptech Ltd. Web Component Development Using Java/Session 11 2

Tell the students that they will be introduced to Expression Language (EL). They will learn the use of EL to access implicit objects and operators in the JSP page. In this session, they will learn how to create static method and tag library descriptor and how to modify deployment descriptor using EL. Explain how to access EL functions within JSP. The session also discusses concept of boxing and unboxing and also how to coerce a value to string or number type.

11.2 In-Class Explanations

Slides 3 to 5

Let us understand Expression Language.

Expression Language 1-3

- ❖ Expression Language (EL) is a primary feature of the JSP technology.
- ❖ EL:
 - ❑ Is simple and robust and can handle both expressions and literals, which are constants and are assigned some memory location.
 - ❑ Provides cleaner syntax and is specifically designed for JSP.
 - ❑ Makes possible to easily access application data stored in JavaBeans components.
 - ❑ Is a great help to the page authors in accessing and manipulating the application data without mastering the complexities of the programming language, such as Java and JavaScript.
 - ❑ Can be used to display the generated dynamic content in a table on a Web page. In addition, EL can also be used in HTML tags.
- ❖ Syntax:

`$ {EL expression}`

 where,
 - ❑ \$ indicates the beginning of an expression in EL.
 - ❑ { is the opening delimiter.
 - ❑ Expression identifies the expression.
 - ❑ } is the closing delimiter.

© Aptech Ltd. Web Component Development Using Java/Session 11 3

Expression Language 2-3

- ❖ The code snippet demonstrates the use of the expression language on JSP.

```
<!-- result.jsp -->
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <body>
    <h1>Qualifying Exam Criteria:</h1>
    <br />
    <br />
    <p>Student should atleast score:</p>
    <pre>
      Subject    Marks
      Maths      ${40+30}
      Java       ${40+35}
      C++        ${40+35}
      Database   ${40+35}
    </pre>
    <br />
    <p>You're accessing the Website on: ${header["user-agent"]}</p>
  </body>
</html>
```

© Aptech Ltd. Web Component Development Using Java/Session 11 4

Expression Language 3-3

- ❖ Output:

Subject	Marks
Maths	70
Java	75
C++	75
Database	75

You're accessing the website on: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:38.0) Gecko/20100101 Firefox/38.0.

© Aptech Ltd. Web Component Development Using Java/Session 11 5

Use slides 3 to 5 explain Expression Language.

Tell the students that The JSP Standard Tag Library (JSTL) expert group and JSP 2.0 expert group at the Java Community Process developed the JSP expression language.

The syntax to use EL in HTML tags is as follows:

```
$ {EL_expression}
```

Here, `EL_expression` specifies the expression itself. The most common operators in JSP EL are `.` and `[]`. These two operators allow you to access various attributes of Java Beans and built-in JSP objects. For example, the JSP expression language allows a page author to access a bean using simple syntax such as `$ {name}`.

An expression language makes it possible to easily access application data stored in JavaBeans components. JSP EL permits the user to create arithmetic and logical expressions. Explain that the user can use integers, floating point numbers, strings, built-in constants such as true and false for boolean values, and null within an EL expression.

With the help of EL, page authors can use simple expressions to dynamically access data from JavaBeans components.

EL allows the user to use simple expressions to perform the following tasks:

- Dynamically read application data stored in JavaBeans components and implicit objects
- Dynamically write data, such as user input to JavaBeans components
- Invoke arbitrary static and public methods
- Dynamically perform arithmetic operations

Then, explain the code snippet that demonstrates the use of the EL on the JSP page. Then, explain how to use EL. Tell them EL can be used in two ways in a JSP page.

1. An attribute value in standard and custom tags. For example, `<jsp:include page="${location}" />`
2. As output with HTML tag. For example, `<h1> User Name: ${username} </h1>`

In-Class Question:

After you finish explaining EL and its features, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which operators are used to access the JavaBean and JSP objects using EL.

Answer:

. and []

Slides 6 and 7

Let us understand request headers and parameters.

Request Headers and Parameters 1-2

- Several implicit objects are available for the easy access through EL .

param

- Returns a value that maps a request parameter name to a single string value.
- The code snippet demonstrates the use of EL to read parameters.

```
<!-- If the request parameter name is null or an empty string,
this snippet returns true. -->
${empty param.Name}
```

paramvalues

- Returns an array of values, which is mapped to the request parameters from client.
- The code snippet demonstrates the use of EL to read parameter values from an array.

```
<!-- Takes the address line as the input -->
<br>Address Line 1: ${paramValues.address[0]}
<br>Address Line 2: ${paramValues.address[1]}
```

© Aptech Ltd. Web Component Development Using Java/Session 11 6

Request Headers and Parameters 2-2

header

- Returns a request header name and maps the value to single string value.
- The code snippet demonstrates the use of header.

```
<!--returns the host as a header name -->
${header["host"]}
```

headerValues

- Returns an array of values that is mapped to the request header.
- Example: <!--Returns the multiple cookies with the same name -->
\${headerValues.name}

cookie

- Returns the cookie name mapped to a single cookie object.
- Example: <!--Returns the value of the cookie -->
\${cookie.name.value}

initParam

- Returns a context initialization parameter name, which is mapped to a single value.

© Aptech Ltd. Web Component Development Using Java/Session 11 7

Use slides 6 and 7 to explain request headers and parameters.

Tell the students that the request parameters are sent from the browser along with the request. Request parameters are typically sent as part of the URL or as part of the body of an HTTP request. If the browser sends an HTTP GET request, the parameters are included in the query string in the URL. If the browser sends an HTTP POST request, the parameters are included in the body part of the HTTP request.

The request headers are name and value pairs sent by the browser along with the HTTP request.

The request headers contain information about what browser software is being used, what file types the browser is capable of receiving etc.

Several implicit objects are available that allow easy access to the following objects:

- **param**: Maps a request parameter name to a single value. For example to access a parameter named product, use the EL `${param.product}` or `${param["product"]}` ;
- **paramValues**: Maps a request parameter name to an array of values.
- **header**: Maps a request header name to a single value. For example, to access a header named user-agent, use the EL `${header.user-agent}` or `${header["user-agent"]}` .
- **headerValues**: Maps a request header name to an array of values.
- **cookie**: Maps a cookie name to a single cookie.
- **initParam**: Maps a context initialization parameter name to a single value.

Following code snippet shows the JSP page asking the username and color. The values from the request are fetched using `param` variable of EL.

user.jsp

```
<html>
. . .
<body>
<form action="display.jsp">
    Student User Name: <input type="text" name="username" /><br>
    Student Choice Color:<input type="text" name="color" /><br>
    <input type="submit" value="Submit Details!!"/>
</form>
</body>
</html>
```

display.jsp

```
<html>
. . .
<body>
    Student User Name is ${ param.username } <br>
    Student Choice Color is ${ param.color }
</body>
</html>
```

Slides 8 and 9

Let us understand scoped variables.

Scoped Variables 1-2

- The four scopes for implicit objects in JSP are as follows:

pageScope <ul style="list-style-type: none"> It returns page-scoped variable names, which are mapped to their values. It is accessible from the JSP page that creates the object. Example: <code><!-- Accesses the page-scoped attribute, book--> \${pageScope.book}</code> 	requestScope <ul style="list-style-type: none"> It provides access to the attributes of request object. It returns request-scoped variable names, which are mapped to their values. It is accessible from Web components handling a request that belongs to the session. Example: <code><!-- Returns the value of the request-scoped attribute, name.--> \${requestScope.student.name}</code>
--	---

© Aptech Ltd. Web Component Development Using Java/Session 11 8

Scoped Variables 2-2

sessionScope <ul style="list-style-type: none"> It returns session-scoped variable names, which are mapped to their values. It is accessible from Web components handling a request that belongs to the session. Example: <code><!-- Returns the value of the numberofPages property of the session-scoped attribute named book. --> \${sessionScope.book.numberOfPages}</code> 	applicationScope <ul style="list-style-type: none"> It returns application-scoped variable and maps the variable name to their values. Example: <code><!-- Checks the value of application-scoped attribute, booklist --> \${applicationScope.book.list == null}</code>
--	---

© Aptech Ltd. Web Component Development Using Java/Session 11 9

Use slides 8 and 9 to explain Scoped Variables. The EL enhances this facility by further supporting the retrieval of the stored objects as scoped variables. The term scoped variable means that the variable is confined to the mentioned context only. The `pageScope`, `requestScope`, `sessionScope`, and `applicationScope` implicit objects are used to retrieve data from request, session, page, and application scopes.

Then, explain the different types of scope variables and their features as explained in the slides. Tell them that you can access variables stored in a specific scope. For example, if you know that the name scoped variable resides in session scope, the expression `${sessionScope.name}` is equivalent to `${name}`.

Tips:

EL scans all the scopes when looking out for a JavaBean name given in the expression. The scopes are scanned in the order: page, request, session, and application. However, the developer can override the search order by explicitly specifying the scope containing the bean. For example, to retrieve the job profile of the `employee` bean located in the session scope, the statement will be: `${sessionScope.employee.job}`

Slide 10

Let us understand page context.

Page Context

- ❖ The `pageContext` implicit object defines the context for the JSP page.
- ❖ It provides access to page attributes.
- ❖ It provides Web page information using the following objects:
 - `servletContext`
 - `session`
 - `request`
 - `response`

© Aptech Ltd. Web Component Development Using Java/Session 11 10

Use slide 10 to explain Page Context. Tell the students `PageContext` implicit object is an instance of `javax.servlet.jsp.PageContext`. Using `PageContext`, the user can find attribute, get attribute, set attribute, and remove attribute from the objects accessible on the current JSP page.

`PageContext` can be used to access the objects at different levels:

- JSP Page – Scope: `PAGE_CONTEXT`
- HTTP Request – Scope: `REQUEST_CONTEXT`
- HTTP Session – Scope: `SESSION_CONTEXT`
- Application Level – Scope: `APPLICATION_CONTEXT`

Explain them that `PageContext` object provides Web page information using the following objects:

➤ **`servletContext`**

The `servletContext` object specifies the servlet of the JSP page and the Web components contained in the same application. It provides methods that a servlet uses to communicate with its servlet container.

➤ **`session`**

The `session` object represents the session created for the client to send requests. It is helpful in maintaining the client state where series of requests are inter-related.

➤ **`request`**

The `request` object represents the request accepted by the JSP page from a client and information of the request.

➤ **response**

The `response` object represents the response sent to the client by a JSP page. The response contains the data passed between a client and servlet.

Some of the methods of `PageContext` are as follows:

- `Object findAttribute (String AttributeName)` : Searches for the specified attribute in Page, Request, Session and Application. When no attribute found at any of the level, it returns NULL.
- `Object getAttribute (String AttributeName, int Scope)` : Looks for an attribute in the specified scope.
- `void removeAttribute(String AttributeName, int Scope)`: Remove an attribute from a given scope.
- `void setAttribute(String AttributeName, Object AttributeValue, int Scope)` : writes an attribute in a given scope.

Following code snippet demonstrates how to store user details in the session scope using `PageContext`.

```
<html>
<body>
<form action="storevalue.jsp">
    Enter User-Id: <input type="text" name="uid"><br>
    <input type="submit" value="User Details">
</form>
</body>
</html>
```

storevalue.jsp

```
<body>
    <% String id=request.getParameter("uid");
        pageContext.setAttribute("UserName", id,
        PageContext.SESSION_SCOPE);
    </body>
```

Slide 11

Let us understand EL operators.

EL Operators

- ❖ EL helps in easy access of application data stored in JavaBeans components.
- ❖ Following table shows all operators used by the JSP EL.

Category	Operators
Variable	. and []
Arithmetic	+, - (binary), *, / and div, % and mod, - (unary)
Conditional	A ? B : C
Relational	==, eq, !=, ne, <, lt, >, gt, <=, le, >=, ge
Logical	and, &&, or, , not, !
Empty/Null checking	empty

© Aptech Ltd. Web Component Development Using Java/Session 11 11

Use slide 11 to explain EL operators. Tell the students that EL supports most of the arithmetic and logical operators supported by Java. Then, explain the list of most frequently used operators mentioned on the slide.

Tell them that all of the binary arithmetic operations prefer `Double` values and all of them will resolve to `0` if either of their operands is `null`. The operators `+`, `-`, `*`, and `%` will try to coerce their operands to `Long` values if they cannot be coerced to `Double`. Like binary arithmetic operations, all of the relational operators prefer `Double` values but will make do with `Long` values if the operands cannot be converted to `Double`. The logical operators prefer to work with `Boolean` operands. You can use the `empty` operator to see if a value is either `null` or an empty string (""). That operator comes in handy when you are interpreting request parameters.

Then, tell them about operator precedence:

- [] .
- ()
- - (unary) not ! empty
- * / div % mod
- + - (binary)
- < > <= >= lt gt le ge
- == != eq ne
- && and
- || or =

The operators are listed from left to right and top to bottom according to precedence; for example, the `[]` operator has precedence over the `.` operator.

Slides 12 and 13

Let us understand EL arithmetic operators.

EL Arithmetic Operators 1-2

- ❖ Operators are used to perform different arithmetic, relational, and logical operations.
- ❖ Dot operator (.) or [] is used to access value of a variable.
- ❖ The EL supports the following arithmetic operators:

- ❖ An arithmetic statement written in JSP EL may contain more than one operator.
- ❖ The EL arithmetic operators accept strings that can be converted into numbers as parameters. Thus, the expression \${"2"+"2"} will evaluate to output 4.

© Aptech Ltd. Web Component Development Using Java/Session 11 12

EL Arithmetic Operators 2-2

- ❖ The code snippet demonstrates the use of EL with arithmetic operators.

```
<%--Following code illustrates how you can use  
the math operators of the EL --%>  
<html>  
${2+2}  
${3-1}  
${2 * 2}  
${10/2}  
${10 div 2}  
${10 % 9}  
${10 mod 9}  
</html>
```

- ❖ The output of this code is : 4 2 4 5.0 5.0 1 1

© Aptech Ltd. Web Component Development Using Java/Session 11 13

Use slide 12 and 13 to explain EL arithmetic operators. Tell them that the EL arithmetic operators accept strings that can be converted into numbers as parameters. Thus, the expression \${ "2"+ "2" } will evaluate to output 4.

Then, explain them the code snippet provided on slide 13 for evaluating expression using EL arithmetic operators.

Tips:

The . operator is similar to the Java. operator, but instead of invoking methods, you access bean properties; for example, if you have a **Name** bean stored in a scoped variable named name and that bean contains `firstName` and `lastName` properties, you can access those properties like this:

First Name: <c:out value='\${name.firstName}' />

Last Name: <c:out value='\${name.lastName}' />

You can also use the [] operator to access bean properties; for example, the preceding code fragment could be rewritten like this:

```
First Name: <c:out value='${name['firstName']}' />  
Last Name: <c:out value='${name['lastName']}' />
```

In-Class Question:

After you finish explaining Arithmetic operators, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



In EL, which operator accepts strings that can be converted into numbers as parameters?

Answer:

Arithmetic operators

Slide 14

Let us understand EL relational operators.

EL Relational Operators

- Relational operators are used to make comparisons against other values, such as boolean, string, integer, or floating point literals.
- Various relational operators used in Expression Language are as follows:

```

graph LR
    Equal == --> Inequality != --> Lesser <
    Lesser --> Greater >
    Greater --> LessEqual <=
    LessEqual --> GreaterEqual >=
    style GreaterEqual fill:#f08080,stroke:#000
  
```

- The code snippet demonstrates the EL with relational operators.

```

<!--comparing numbers-->
4 > '3' ${4 > '3'}<br/>
'4' > 3 ${'4' > 3}<br/>
'4' > '3' ${'4' > '3'}<br/>
4 >= 3 ${4 >= 3}<br/>
4 <= 3 ${4 < 3}<br/>
4 == 4 ${4 == 4}<br/>
  
```

© Aptech Ltd. Web Component Development Using Java/Session 11 14

Use slide 14 to explain EL relational operators. Tell the students that relational operator is used to check the values returned by two expressions or operands. In this operator, comparisons can be done with other values like boolean, string, integer, or floating point literals.

Example for relational operators are shown in the following table:

Operators	Expression	Result
Equal	<code> \${11.0==11}</code>	true
Equal	<code> \${11.0 eq 11}</code>	true
Not Equal	<code> \${((30*10)!= 300)}</code>	true
Not Equal	<code> \${5 ne 5}</code>	false
Greater than or equal	<code> \${5.2>=2}</code>	true
Greater than or equal	<code> \${5.4 ge 4}</code>	true
Less than	<code> \${1<5}</code>	true
Less than	<code> \${5 lt 9}</code>	true
Less than or equal	<code> \${8 <= 14}</code>	true
Less than or equal	<code> \${10 le 3}</code>	true

Slide 15

Let us understand EL logical operators.

EL Logical Operators

- ❖ The logical operators supported by EL are as follows:

and, &&
(Logical AND)

or, ||
(Logical OR)

! or not
(Boolean complement)

- ❖ The code snippet demonstrates the use of EL with logical operators.

```

<!--Test the condition-->
<c:if test="${(guess >= 10) && (guess <= 20) }">
    <b>You're in the range!</b><br/>
</c:if>
<c:if test="${(guess < 10) || (guess > 20) }">
    <b>Try again!</b><br/>
</c:if>
```

© Aptech Ltd. Web Component Development Using Java/Session 11 15

Use slide 15 to explain EL logical operators. Tell the students that Logical operators are used with two operands that returns either a ‘true’ or ‘false’ value.

Example for logical operators are shown in the following table:

Operators	Expression	Result
And	<code> \${true and true}</code>	true
And	<code> \${true && false}</code>	false
Or	<code> \${true or true}</code>	true
Or	<code> \${true false}</code>	true
Not	<code> \${not true}</code>	false
Not	<code> \${!false}</code>	true

Slide 16

Let us understand EL empty operators

EL Empty Operators

- ❖ It is a prefix operation that can be used to determine whether the value is null or empty.
- ❖ It returns true if the string is empty.
- ❖ The string is said to be empty if it contains no character.
- ❖ If the string is not empty, then empty operator returns false.

❖ The code snippet demonstrates the use of empty operators.

```
<% -- this code returns true if the string is empty
and string "sometext" --%>
<b>
empty "" ${empty ""}<br/>
empty "sometext" ${empty "sometext"}<br/>
</b>
```

© Aptech Ltd. Web Component Development Using Java/Session 11 16

Use slide 16 to explain EL empty operators. Tell the students that empty operator is used to test whether an identifier is null or doesn't exist.

The `empty` operator returns `true` if the string is empty. The string is said to be empty if it contains no character. If the string is not `empty`, then `empty` operator returns `false`.

Then, explain them the use of `empty` operator to test null string.

The following code snippet shows the testing of a string value using `empty` operator.

```
<p> Empty Operator Example</p>
The Value for the Empty operator is: ${empty "txxt"}
```

Result:

Empty Operator Example

The Value for the Empty operator is: false

Empty operator checks finds the string so it is not "null", so returns "false".

Slides 17 to 20

Let us understand EL dot operators.

EL Dot Operators 1-4

- ❖ It is used to access attribute values of JavaBean and map values within the EL.
- ❖ The code to the left of the operator must specify a JavaBean or a map.
- ❖ The code to the right of the operator must specify a JavaBean or a map key.
- ❖ The property name follows the conventions of Java identifiers.

Syntax:

```
 ${mapObject.keyName}
```

- ❖ The EL allows to replace dot notation with array notation (square brackets).
- ❖ Example: \${param.header} can be replaced with \${param["header"]}
- ❖ The advantages of array notation are as follows:
 - Can be used to access elements of an array or a list by specifying an index.
 - Allows to use values as property names.

© Aptech Ltd. Web Component Development Using Java/Session 11 17

EL Dot Operators 2-4

- ❖ The code snippet demonstrates the use of dot operator.

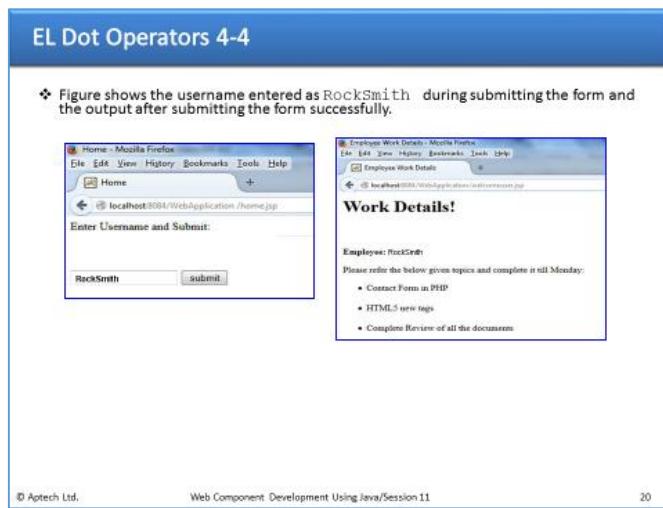
```
<!-- home.jsp -->
<html>
  <head> . . . </head>
  <body>
    <p>Enter Username and Submit:</p><br /><br />
    <form action="welcomeuser.jsp" method="post">
      <input name="txtval" type="text" size="20"
      maxlength="60" placeholder="Enter Username..." required/>
      <input name="sbtName" type="submit" value="submit" />
    </form>
    <%
    // setAttribute(name,object) binds an object to a given attribute
    application.setAttribute("PHP", "Contact Form in PHP");
    application.setAttribute("HTML5", "HTML5 new tags");
    application.setAttribute("Review", "Complete Review of all
    the documents");
    %>
  </body>
</html>
```

© Aptech Ltd. Web Component Development Using Java/Session 11 18

EL Dot Operators 3-4

```
<!-- welcomeuser.jsp -->
<%@page contentType="text/html"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  . . .
  <body>
    <h1>Work Details!</h1> <br /><br />
    <b>Employee: ${param.txtval}</b> <br />
    <p>Please refer to the given topics and
    complete it till Monday:</p>
    <ul>
      <li>${applicationScope.PHP}</li> <br />
      <li>${applicationScope.HTML5}</li> <br />
      <li>${applicationScope.Review}</li> <br />
    </ul>
  </body>
</html>
```

© Aptech Ltd. Web Component Development Using Java/Session 11 19



Use slides 17 to 20 to explain EL dot operators. Tell the students that (.) operator within EL is used to access properties of bean and map values.

To access JavaBean property

JSP EL Dot operator is used to get the attribute values. For example, \${firstObj.secondObj} where, firstObj can be EL implicit object or an attribute in page, request, session, or application scope.

To access the employee object address stored in the request scope, the EL will be
\${requestScope.employee.address}

Tell them that the implicit requestScope is just a Map of the request scope **attributes**, not the request object itself.

To access collection object

[] operator is more powerful than . operator. We can use it to get data from List and Array too.

The expression language lets you replace dot notation with array notation (square brackets). So, for example, you can replace \${name.property} with \${name["property"]}

The second form is rarely used with bean properties. However, it does have two advantages. First, it lets you compute the name of the property at request time. With the array notation, the value inside the brackets can itself be a variable; with dot notation the property name must be a literal value.

The JSP 2.0 expression language lets you access different types of collections in the same way: using array notation. For instance, if attributeName is a scoped variable referring to an array, List, or Map, you access an entry in the collection with the following:

\${attributeName[entryName]}

Consider, if scoped variable is an array, the entry name is the index and the value is obtained with the syntax, array[index]. For example, if customerNames refers to an array of strings,

\${customerNames[0]} will display the entry from the first element.

Now, consider that the scoped variables is an object that implements the Map interface, the entry

name is the key and the value is obtained with the `Map.get(key)`. For example, if statecapitals is the `HashMap` object, then `${stateCapitals["maryland"]}` would return some value on the specified key.

The same can be return as `${stateCapitals.maryland}` replacing the array notation with dot notation.

Then, discuss with them the advantages of using Array notations over Dot notation.

Tell them that the array notation lets you choose the key at request time, whereas the dot notation requires you to know the key in advance. Dot notation is used only when the key is in a form of a Java variable name. Array notation operator can be used to access elements of an array or a list by specifying an index. That is the value inside the brackets can itself be a variable in array notation, while with dot notation the property name must be a literal value.

Then, explain the code snippet mentioned on slides 18 and 19 to explain the use of dot operator.

Slides 21 and 22

Let us understand creating static methods.

Creating 'static' Methods 1-2

- ❖ The `static` java methods can be called within the EL expression.
- ❖ To access the function using EL, the function must be implemented as a `static` function in a Java class.
- ❖ A user can define many functions in a single class.
- ❖ After defining the functions, a user need to map the function name with EL using a Tag Library Descriptor (TLD) file.
- ❖ Syntax:
`ns:funcName(arg1, arg2, . . .)`

where,

- ❑ `ns` refers to name space and function name. Name space is generally a class or a tag library or a function name to access a static method in some class.

© Aptech Ltd. Web Component Development Using Java/Session 11 21

Creating 'static' Methods 2-2

- ❖ Following figure demonstrates the `static` function.

```
package mypackage;
public class MyFunctions {
    public static double
        average(double [] values){
        double dblSum=0.0;
        int iCount = values.length();
        for (int i=0;i<iCount;i++)
            dblSum+=values[i];
        return dblSum/iCount;
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 11 22

Use slides 21 and 22 to explain creating static methods. Tell them to access the function using EL, the function must be implemented as a `static` function in a Java class. You can define many functions in a single class. After defining the functions, you need to map the function name with EL using a Tag Library Descriptor (TLD) file.

Then, discuss with them how to create a static function as demonstrated on slide 22.

In-Class Question:

After you finish explaining the creation of static methods, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which method can be called within the EL expressions?

Answer:

Static Java methods

Slides 23 and 24

Let us understand creating tag library descriptor.

Creating Tag Library Descriptor 1-2

- ❖ A TLD file uses XML syntax to map the name of functions defined in a class with EL.
- ❖ In the `<function>` tag, in a tag library descriptor file, user need to mention the name of the function using element `<name>`.
- ❖ A user also need to mention the class in which the function is defined using `<function-class>` element and the signature of the function in the TLD file using `<function-signature>` element.
- ❖ Then save this TLD file in the `/WEB-INF/tlds` folder, where `tlds` is a user-created folder.

© Aptech Ltd. Web Component Development Using Java/Session 11 23

Creating Tag Library Descriptor 2-2

- ❖ Following figure depicts EL function configuration in the TLD.

```

<function>
  <name>average</name>
  <function-class>mypackage.
    MyFunctions
  </function-class>
  <function-signature>
    double average
    (double[])
  </function-signature>
</function>

```

© Aptech Ltd. Web Component Development Using Java/Session 11 24

Use slide 23 and 24 to explain Tag Library Descriptor used for configuring the function in the EL. Tell the students that tag library allows the user to group tags with related functionality. A tag library uses a tag library descriptor (TLD) file that identifies the tag extensions and connects them to their Java classes. TLD files have the file extension `.tld` and are written in XML notation.

In the `<function>` tag in a tag library descriptor file, you need to mention the name of the function using element `<name>`. Also, you need to mention the class in which the function is defined using `<function-class>` element and the signature of the function in the TLD file using `<function-signature>` element.

Tips:

The `.tld` file is saved in the `/WEB-INF/tlds` folder, where `tlds` is a user-created folder.

Slides 25 and 26

Let us understand modifying deployment descriptor.

Modifying Deployment Descriptor 1-2

- ❖ The default mode for JSP version 1.2 technology or before is to ignore EL expressions.
- ❖ The default mode for JSP pages delivered with JSP version 2.0 technology is to evaluate EL expressions.
- ❖ Setting the value of the `<el-ignored>` element in the deployment descriptor can explicitly change the default mode.
- ❖ The `<el-ignored>` element is a sub element of `<jsp-property-group>`.
- ❖ It has no sub elements.
- ❖ Its valid values are true and false.
- ❖ In deployment descriptor, `web.xml` file, declare as follows:

Syntax:

```
<el-ignored>false</el-ignored>
```

where,

- ❑ true indicates that EL expressions will be ignored.
- ❑ false indicates that EL expressions will be enabled for interpretation by servlet container.

© Aptech Ltd. Web Component Development Using Java/Session 11 25

Modifying Deployment Descriptor 2-2

- ❖ The code snippet shows the deployment descriptor, `web.xml` which has to be modified in the following ways:

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    <jsp-config>
        <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
            <el-ignored>false</el-ignored>
        </jsp-property-group>
    </jsp-config>
</web-app>
```

- ❑ `<jsp-config>`:Includes JSP configuration, such as interpretation of tag library and property information.
- ❑ `<jsp-property-group>`:Defines a set of properties that applies to a set of files representing the JSP pages.
- ❑ `<url-pattern>`:Specifies that JSP properties defined in `<jsp-property-group>` to specific JSP pages, *.jsp indicates that these apply to all JSP pages.
- ❑ `<el-ignored>`:Enables interpretation of JSP EL in JSP pages.
- ❑ `<scripting-enabled>`:Allows JSP scripting.

© Aptech Ltd. Web Component Development Using Java/Session 11 26

Use slides 25 and 26 to modify deployment descriptor. Tell the students that a deployment descriptor describes how a component, module, or application should be deployed. It directs a deployment tool to deploy a module or application with specific container options, security settings, and describes specific configuration requirements.

Tell them that the earlier JSP technology prior to JSP 2.0, the default mode for EL expression was set to ignore. However, with JSP 2.0 specification, the EL is activated by default.

To deactivate the EL in multiple JSP pages, you use the `el-ignored` subelement of the `jsp-property-group` `web.xml` element to designate the pages in which the expression language should be ignored.

Then, discuss with them the deployment descriptor, to modify the settings for EL which show the following statements:

```
<jsp-config>  
  
<jsp-property-group>  
    <url-pattern>/legacy/*.jsp</url-pattern>  
    <el-ignored>true</el-ignored>  
</jsp-property-group>  
  
</jsp-config>
```

Within the `jsp-config` tag, the developer can declare zero or more `<jsp-property-group>` elements. The `<el-ignored>` tag has the value `false`, indicating that EL is executed. If set to `true`, any EL used on the JSP pages in the project is ignored.

This setting may be useful when pages from older project with EL-like text are used.

Tell them to disable EL evaluation in an individual page, supply `false` as the value of the `isELEnabled` attribute of the `page` directive, as follows:

```
<%@ page isELEnabled="false" %>
```

Tips:

Note that the `isELEnabled` attribute is new in JSP 2.0 and it is an error to use it in a server that supports only JSP 1.2 or earlier. So, you cannot use this technique to allow the same JSP page to run in either old or new servers without modification. Consequently, the `jsp-property-group` element is usually a better choice than the `isELEnabled` attribute.

Slide 27

Let us understand accessing EL functions within JSP.

Accessing EL Functions within JSP

- ❖ To access the function created in a TLD file using a JSP file, you need to import the TLD file using the `taglib` directive.
- ❖ In the directive statement, you need to mention a prefix for the tags and the location of the TLD file.
- ❖ After importing the TLD file, you can access the function using an EL expression.
- ❖ For the taglib directive, syntax declares as follows:

Syntax:

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

where,

- prefix is the prefix to be used for the tags defined in the TLD file.
- path is the location of the TLD file.

- ❖ For accessing the function:

```
<${prefix:functionName(arguments)}%>
```

- ❖ The code snippet shows how to access EL function in JSP.

```
<%@ taglib prefix="fn" uri="/WEB-INF/tlds/functions"%>
Average of the values is : <${fn:average(values)}%>
```

© Aptech Ltd. Web Component Development Using Java/Session 11 27

Use slide 27 to explain accessing of EL functions within JSP.

Tell the students that to use a function in a JSP page, user should use a `taglib` directive to import the tag library containing the function.

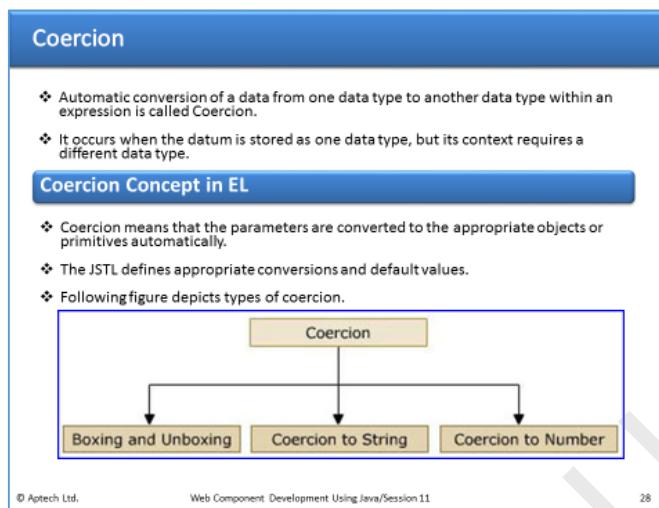
To access the function created in a TLD file using a JSP file, you need to import the TLD file using the `taglib` directive. In the directive statement, you need to mention a prefix for the tags and the location of the TLD file as:

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

Then, explain them the code snippet to access the EL function in JSP.

Slide 28

Let us understand coercion.



Use slide 28 to explain Coercion. Tell them Coercion means that the parameters are converted to the appropriate objects or primitives automatically. The JSTL defines appropriate conversions and default values. For example, a string parameter from a request will be coerced to the appropriate object or primitive.

If a parameter that represents the month is passed in the request as a string, the value of the month variable will be correct because the string will be coerced to the correct type when used.

Then, tell them about the type of coercion as shown in the figure presented on the slide.

Slide 29

Let us understand boxing and unboxing.

Boxing and Unboxing

- ❖ Boxing converts values of primitive type to corresponding values of reference type.
- ❖ Unboxing converts values of reference type to corresponding values of primitive type.

The precise rules for boxing

- A) If i is a boolean value, then boxing conversion converts i into a reference r of class and type Boolean, such that $r.value() == i$.
- B) If i is a byte value, then boxing conversion converts i into a reference r of class and type Byte, such that $r.value() == i$.

The precise rules for unboxing

- A) If r is a Boolean reference, then unboxing conversion converts r into a value v of type boolean, such that $r.value() == v$.
- B) If r is a Byte reference, then unboxing conversion converts r into a value v of type byte, such that $r.value() == v$.
- C) If r is a Character reference, then unboxing conversion converts r into a value v of type char, such that $r.value() == v$.

© Aptech Ltd. Web Component Development Using Java/Session 11 29

Use slide 29 to explain Boxing and Unboxing. Tell the students that boxing is the process, in which the primitive types is converted into their corresponding object wrapper classes. Similarly, unboxing is a process by which the value of the object is extracted automatically from a wrapper class

Then, discuss with them the rules for boxing and unboxing followed in JSP.

In-Class Question:

After you finish explaining Boxing and Unboxing, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



In which process, the primitive types is converted into their corresponding object wrapper classes?

Answer:

Boxing.

Slides 30 to 32

Let us understand coercing a value to string and number.

Coercion to String

- ❖ The rule to coerce a value to String type is as follows:
 - If A is String, return A.
 - If A is null, return "".
 - If A.toString() throws exception, return error.
Otherwise, return A.toString().

© Aptech Ltd. Web Component Development Using Java/Session 11 30

Coercion to Number 1-2

- ❖ The rule to coerce a value to number type is as follows:
 - If A is null or "", return 0.
- ❖ A is character and is converted to short, apply the following rules:
 - If A is Boolean, return error.
 - If A is number type, return A.

© Aptech Ltd. Web Component Development Using Java/Session 11 31

Coercion to Number 2-2

- ❖ A is number, coerce occurs quietly to type N using the following algorithms:
 - If N is BigInteger
 - If A is BigDecimal, return A.toBigInteger()
 - Otherwise, return BigInteger.valueOf(A.longValue())
 - If N is BigDecimal
 - If A is a BigInteger, return new BigDecimal(A)
 - Otherwise, return new BigDecimal(A.doubleValue())
 - If N is Byte, return new Byte(A.byteValue())
 - If N is Short, return new Short(A.shortValue())
 - If N is Integer, return new Integer(A.integerValue())
 - If N is Long, return new Long(A.longValue())
 - If N is Float, return new Float(A.floatValue())
 - If N is Double, return new Double(A.doubleValue())
 - Otherwise, return error

© Aptech Ltd. Web Component Development Using Java/Session 11 32

Use slides 30 to 32 to explain Coercing a value to string and number. Discuss with them the rules to coerce a value of string type as mentioned on the slide. Then, discuss with them how a value is coerced into an appropriate number type as mentioned on slides 31 and 32.

Slide 33

Let us summarize the session.

Summary

- ❖ EL is simple and robust. It can handle both expressions and literals, which are constants and are assigned some memory location.
- ❖ EL is a great help to the page authors in accessing and manipulating the application data without mastering the complexities of the programming language such as Java and JavaScript.
- ❖ JSP implicit objects are a standard set of classes. The user creates an instance of an implicit object to use available methods and variables.
- ❖ Operators are used to perform different arithmetic, relational, and logical operations. Dot operator (.) or [] is used to access value of a variable. Various operators used in Expression Language are arithmetic operators, relational operators, logical operators, empty operators, and dot operators.
- ❖ In Expression language, the static java methods can be called within the EL expression. To access the function using EL, the function must be implemented as a static function in a java class.
- ❖ The TLD file uses XML syntax to map the name of functions defined in a class with EL. Setting the value of the <el-ignored> element in the deployment descriptor can explicitly change the default mode.
- ❖ The accessing of the function created in a TLD file using a JSP file is possible by importing the TLD file using the taglib directive.
- ❖ Coercion means that the parameters are converted to the appropriate objects or primitives automatically. Coercion is an implicit type conversion.

© Aptech Ltd. Web Component Development Using Java/Session 11 33

In slide 33, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

11.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the Java Standard Tag Library (JSTL) that is explained with the next session.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 12 – JavaServer Pages Standard Tag Library

12.1 Pre-Class Activities

Familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

12.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the concept and need for JSTL
- List the advantages of using JSTL
- Describe the different tag libraries available in JSTL
- Explain how to configure JSTL library in NetBeans
- Explain general purpose tags
- Explain decision-making in the tags
- Explain iteration tags in the core tag library
- Explain the different tags available in the SQL tag library

12.1.2 Teaching Skills

To teach this session successfully, you should be aware of JavaServer Pages Standard Tag Library (JSTL). Familiarize yourself with the different tag libraries available in JSTL. You should know how to configure JSTL library in NetBeans. You should also aware yourself with the usage of general purpose tags, decision-making tags, iteration tags, and different tags available in SQL tag library.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❖ Explain the concept and need for JSTL
- ❖ List the advantages of using JSTL
- ❖ Describe the different tag libraries available in JSTL
- ❖ Explain how to configure JSTL library in NetBeans
- ❖ Explain general purpose tags
- ❖ Explain decision-making in the tags
- ❖ Explain iteration tags in the core tag library
- ❖ Explain the different tags available in the SQL tag library

© Aptech Limited. Web Component Development Using Java/Session 12 2

Tell the students that they will be introduced to JavaServer Pages Standard Tag Library (JSTL). Explain different tag libraries available in JSTL. Explain how to configure JSTL library in NetBeans. In this session, they will learn general purpose tags, decision-making tags and iteration tags. Finally discuss the different tags available in SQL tag library.

12.2 In-Class Explanations

Slide 3

Let us understand JavaServer Pages Standard Tag Library (JSTL).

The slide has a blue header bar with the word 'Introduction'. Below it is a cartoon illustration of a person sitting at a desk with a computer monitor, pointing at it. To the right of the illustration is a purple callout box containing the text: 'How a Web designer can design a Web page to display a dynamic list in a Shopping cart site with selected or removed products?'. The main content area contains the following text:

- ❖ To help Web designers, Sun developed a pre-defined tag library.
- ❖ The library contains tags related to the core common functionalities performed in Java applications.
- ❖ The pre-defined tag library is known as **JavaServer Pages Standard Tag Library (JSTL)**.
- ❖ JSTL:
 - ❑ Helps to reuse standard tags that work in the similar manner in every Java Web application.
 - ❑ Allows programming using tags rather than scriptlet code.

At the bottom left is the copyright notice: © Aptech Limited. At the bottom center is the page number: 3. At the bottom right is the text: Web Component Development Using Java/Session 12.

Use slide 3 to explain JavaServer Pages Standard Tag Library (JSTL). Discuss the requirement of a Web designer to generate a dynamic list displaying user-selected products on the Web page. To display such a dynamic list in a cart with selected or removed products, only HTML tags will not be useful. In such a case, the Web designer has to involve himself in coding the scripts, such as iterating through the list elements or displaying data based on a condition. This may not be feasible in all the situations.

To help Web designers, Sun Microsystems developed a pre-defined tag library which contains tags related to the core common functionalities performed in Java applications. The pre-defined tag library is known as JavaServer Pages Standard Tag Library (JSTL) that helps you to reuse standard tags that work in the similar manner in every Java Web application.

Tell the students that JSTL is a collection of JSP tags which provide core functionalities common to most of the JSP applications.

JSTL provides a framework for integrating existing custom tags with JSTL tags.

Main advantages of JSTL are as follows:

- JSTL removes the complexities of JSP by providing tags.
- JSTL tags can be used in various pages.
- Scriptlet tag is not used in JSTL.

Slides 4 and 5

Let us understand designing JSP pages with JSTL.

Designing JSP Pages with JSTL 1-2

- ❖ The roles found in designing and developing a Web applications are as follows:
 - Web Designers**
 - They are involved in creating a view part of the application.
 - The views are basically HTML pages.
 - Web Component Developers**
 - They are responsible for developing the controller of the application.
 - The controller is basically a Servlet written in Java language.
 - Business Component Developers**
 - They are responsible for creating the model for the application.
 - The model is a component, such as a Java class or an Enterprise JavaBean (EJB) that are used to process the business logic of an application.

© Aptech Limited. Web Component Development Using Java/Session 12 4

Designing JSP Pages with JSTL 2-2

- ❖ The benefits of using JSTL are as follows:
 - JSTL help Web designers to integrate the Java technology code into JSP, without the need to write complex scriptlet code.
 - JSTL provides most of the functionality necessary for the development of JSP application.
 - JSTL tags helps to create a program with business logic that saves lots of development time.
 - JSTL is similar to HTML and therefore HTML programmers easily start programming using JSTL.
 - JSTL is expressed in XML-compliant tags and therefore it is easier for HTML generation tools to parse the JSTL code contained within the document.
 - JSTL tags provides a consistent approach to formatting of numbers and strings, and internationalization (I18N) support features in JSP scriptlet code.
 - JSTL provides mechanism that enables the programmer to develop own custom tags.

© Aptech Limited. Web Component Development Using Java/Session 12 5

Use slides 4 and 5 to explain designing JSP pages with JSTL. Tell them that organizations handling large Web applications need separate job roles to handle different parts of the development.

Then, discuss the roles of different stakeholders involved in Web application development as mentioned on slide 4.

Tell them that the use of tag libraries, such as JSTL helps people involved in the application development to focus on their area of expertise, such as Web designers, who are involved in designing Web pages. The JSTL tag library help Web designers to integrate the Java technology code into JSP, without the need to write complex scriptlet code.

Then, describe the advantages of using JSTL in the development of JSP applications as mentioned on slide 5.

In-Class Question:

After you finish designing JSP pages with JSTL, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Who is responsible for developing the controller of the application?

Answer:

Web Component Developers

Slides 6 to 9

Let us understand JSTL tag libraries.

JSTL Tag Libraries 1-3

- ❖ JSTL provide various tag libraries that can be used for many functionalities.
- ❖ Following figure shows the JSTL tag libraries.

```

graph LR
    JSTL((J  
S  
T  
L)) --> Core[Core tag library]
    JSTL --> Format[Formatting/  
Internationalization tag library]
    JSTL --> SQL[SQL tag library]
    JSTL --> XML[XML tag library]
  
```

© Aptech Limited. Web Component Development Using Java/Session 12 6

JSTL Tag Libraries 2-3

- ❖ **Core Tag Library**
 - ❑ It contain tags for looping, expression evaluation, and basic input/output.
 - ❑ It can be declared by `<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>`.
- ❖ **Formatting/Internationalization (I18N) Tag Library**
 - ❑ It contain tags used to parse the data such as dates and time, based on the current location.
 - ❑ It can be declared by `<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>`.

© Aptech Limited. Web Component Development Using Java/Session 12 7

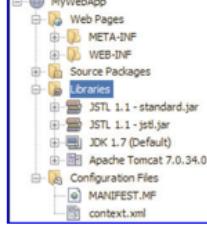
JSTL Tag Libraries 3-3

- ❖ **SQLTag Library**
 - ❑ It contains tags used to access SQL databases.
 - ❑ It provides an interface for executing SQL queries on database through JSP.
 - ❑ It can be declared by `<%@ taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>`.
- ❖ **XML Tag Library**
 - ❑ It contains tags for accessing XML elements.
 - ❑ It can be declared by `<%@ taglib prefix="x" uri="http://java.sun.com/jstl/xml" %>`.

© Aptech Limited. Web Component Development Using Java/Session 12 8

JSTL Library in NetBeans

- ❖ To use the JSTL tags on the JSP page, it is required to include the JSTL library in the Web application.
- ❖ To do so, perform the following steps:
 - ❑ Right-click **Libraries** and select **Add Library**.
 - ❑ Select **JSTL 1.1** under **Available Libraries** and click **Add Library**. The **JSTL 1.1-standard.jar** and **JSTL 1.1-jstl.jar** are added in the project.
 - ❑ Following figure shows the JSTL library added in **MyWebApp** application project created in NetBeans IDE.



© Aptech Limited. Web Component Development Using Java/Session 12 9

Use slide 6 to 9 to explain JSTL tag libraries. Tell the students that there are different types of tag libraries offered by JSTL. JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating existing custom tags with JSTL tags.

- **JSTL Core tags** - <http://java.sun.com/jsp/jstl/core>
- **JSTL Internationalization tags** - <http://java.sun.com/jsp/jstl/fmt>
- **JSTL Function tags** - <http://java.sun.com/jsp/jstl/functions>
- **JSTL Database tags** - <http://java.sun.com/jsp/jstl/sql>
- **JSTL XML tags** - <http://java.sun.com/jsp/jstl/xml>

Explain each type of tag library provided by JSTL.

Following table summarizes the functionalities provided by each category of JSTL tags with their prefix on the JSP page:

Tag	Functionality	Prefix
Core	Variable support	c
	Flow control	
	URL management	
	Miscellaneous	
XML	Core	x
	Flow control	
	Transformation	
I18N	Locale	fmt
	Message formatting	
	Number and date formatting	
Database	SQL	sql
Functions	Collection length	fn
	String manipulation	

Tips:

The JSP specification provides standard tags for use within the JSP pages. Thus, every JSP container must support them. This also ensures that any application using standard tag will work in any JSP complaint container.

The standard tags begin with `jsp:` prefix. However, standard tags have limited functionality, thus the introduction of JSTL and EL reduces the need for standard tags.

Then, explain them how to configure JSTL library in NetBeans IDE.

Slide 10

Let us understand core tag library.

Core Tag Library

- ❖ It provides support for conditional logic, iteration, forward or redirect response, URL, catch exception, and so on.
- ❖ It is the most frequently used JSTL tag library containing core group of tags.
- ❖ The core tag library tags are prefixed with c.
- ❖ Some of the Core JSTL tags include:
 - ❑ General Purpose Tags
 - ❑ Decision-making Tags
 - ❑ Iteration Tags

© Aptech Limited. Web Component Development Using Java/Session 12 10

Use slide 10 to explain core tag library. Core tag library provide support for conditional logic, iteration, forward or redirect response, URL, catch exception, and so on. It is the most frequently used JSTL tag library containing core group of tags. The core tag library tags are prefixed with c.

Slides 11 to 17

Let us understand general purpose tags.

General Purpose Tags 1-7

- ❖ They are used to set, remove, and display variable values created within a JSP page.
- ❖ The core tag library contains tags for getting, setting, and displaying attribute values.
- ❖ The general purpose tags are as follows:

<c:set>

- It assigns a value to a variable in scope.
- **Syntax:**

```
<c:set var = "varName" value = "expression"
       scope = "page/request/ session/ application"/>
```

where,

- ❑ value specifies the expression.
- ❑ var specifies the name of the exported scope to hold the value specified in the tag.
- ❑ scope specifies the scope of variable such as page, request, session, and application. The default scope is page.

- **Example:** <c:set var = "sessionvariable" value = "\${80+8}" scope = "session" />

© Aptech Limited. Web Component Development Using Java/Session 12 11

General Purpose Tags 2-7

<c:remove>

- This is an empty tag used to remove a scoped variable.
- **Syntax:**

```
<c:remove var = "varName" scope =
           "page/request/session/application"/>
```

where,

- ❑ var specifies the name of the variable to be removed.
- ❑ scope specifies the scope of the variable.

- **Example:** <c:remove var="simple" scope="page"/>

© Aptech Limited. Web Component Development Using Java/Session 12 12

General Purpose Tags 3-7

<c:out>

- It is used to evaluate an expression and store the result in the current JspWriter object.
- **Syntax:**

```
<c:out value = "value" escapeXml =
        "boolean" default = "defaultValue"/>
```

where,

- ❑ value specifies the expression.
- ❑ default specifies the default value if the value of the result is null.
- ❑ escapeXml determines that special characters in the result, such as '<', '>', '&', and '' should be converted to their character entity codes. The default value of escapeXml is true.

- **Example:** <c:out value = "\${sessionvariable}">
</c:out>

© Aptech Limited. Web Component Development Using Java/Session 12 13

General Purpose Tags 4-7

<c:catch>

- It provides an exception handling functionality such as try-catch, inside JSP pages without using scriptlets.

• **Syntax:**

```
<c:catch [var="varName"]>
    nested actions
    ...
</c:catch>
```

where,

□ var specifies the name of the variable to be caught.

© Aptech Limited.

Web Component Development Using Java/Session 12

14

General Purpose Tags 5-7

- ❖ The code snippet demonstrates how to catch the exception using <c:catch> tag.

```
<body>
<c:catch var="e">
    100 divided by 0 is
    <:out value="\${100/0}" />
<br />
</c:catch>
```

- ❖ The **catch** tag provides an error handling mechanism for the division operation.
- ❖ The exception raised by dividing the number from 0 is stored in the variable **var**.

© Aptech Limited.

Web Component Development Using Java/Session 12

15

General Purpose Tags 6-7

- ❖ The code snippet demonstrates the core tags used in login application.

```
<!-- welcome.html -->
<body>
<form action="login.jsp">
User Name:<br/>
<input type="text" name="username"/>
<br/><br/>
Password:<br/>
<input type="password" name="password"/>
<br/><br/>
<input type="submit" value="Submit" name="Submit">
</form>
</body>
```

© Aptech Limited.

Web Component Development Using Java/Session 12

16

General Purpose Tags 7-7

```
<!-- login.jsp -->
. .
<body>
    <c:set var="name" value="${param.username}" />
    <c:set var="password" value="${param.password}" />

    Entered Name: <c:out value="${name}" /> <br/> <br/>
    Entered Password: <c:out value="${password}" />

</body>
```

- ❖ The login.jsp page assigns the variables name and password with the values from the request parameters username and password respectively.
- ❖ The out tag outputs the value of name and password respectively.

© Aptech Limited. Web Component Development Using Java/Session 12 17

Use slides 11 to 17 to explain general purpose tags. Explain them the various tags of core tag library.

➤ <c:set> and <c:Remove>

The <c:set> and <c:Remove> tags are used to set and remove the value from the defined scope. The <c:remove> tag removes the variable from the specified scope. User have to specify the scope from which the variable have to be removed, if the scope is not specified, then it will remove the variables from all scopes.

Syntax of <c:set> tag - <c:set var = "varName" value = "expression" scope = "page/request/ session/ application"/>

Syntax of <c:remove> tag - <c:remove var = "varName" scope = "page/request/session/application"/>

Example: to set the value for a variable whose scope will be session, then the <c:set> tag can be used as follows:

```
<c:set var="user" scope="session" value="John" />
```

The tag creates a session-scoped variable named “user”, if the variable is not present in the scope. If the variable or attribute is present and set to NULL, then it will be removed. In case if the developer has to set multiple value then, the following tag can be used:

```
<c:set var="user" scope="session" >
    Sheriff, John, Cowboy
</c:set>
```

To set the value of a bean or map using <c:set>, the developer can set the target attribute as shown:

```
<c:set target="${PetMap}" property="dogName" value="Clover" />
```

Here, the target must be an object of type JavaBean or Map. The `petMap` is the map object, whereas the “dogname” is the key for the value “Clover”.

To remove the variable, the `<c:remove>` tag can be used.

```
<c:remove var="user" scope="session" />
```

Scope is optional, if not mentioned, page scope is default.

➤ `<c:out>`

The `<c:out>` tag displays the output to the user. Attributes of `<c:out>` tag are `value`, `default`, and `escapeXml`. The `out` tag evaluates an expression and outputs the result of the evaluation to the current `JspWriter` object.

The `escapeXml` attribute if set to true, the character conversions listed in the following table is applied:

Character	Character Entity Code
<	<
>	>
&	&
'	'
"	"

➤ `<c:catch>`

The `catch` tag provides a complement to the JSP error page mechanism. It allows page authors to recover gracefully from error conditions that they can control. Actions that are of central importance to a page should not be encapsulated in a catch; in this way their exceptions will propagate instead to an error page. Actions with secondary importance to the page should be wrapped in a catch so that they never cause the error page mechanism to be invoked.

The exception thrown is stored in the variable identified by `var`, which always has page scope. If no exception occurred, the scoped variable identified by `var` is removed if it existed. If `var` is missing, the exception is simply caught and not saved.

The following code snippet shows the use of `<c:catch>` tag:

```
<body>
<c:catch var="e">
100 divided by 0 is <c:out value="${100/0}" />
<br />
</c:catch>
```

Then, explain the login application demonstrating the use of the core tags.

In-Class Question:

After you finish explaining General Purpose tags, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which tag removes the variables from all scopes, if the scope from which the variable have to be removed is not specified?

Answer:

<c:remove> tag

Slides 18 to 22

Let us understand decision-making tags.

Decision-making Tags 1-5

- ❖ JSTL provides decision-making tags to support conditions in a JSP page.
- ❖ These tags are necessary as the contents or the output of the JSP page is often conditional based on the value of the dynamic application data.
- ❖ The two types of decision-making tags are as follows:
 - ❑ <c:if>
 - ❑ <c:choose>

© Aptech Limited. Web Component Development Using Java/Session 12 18

Decision-making Tags 2-5

<c:if>

- The tag is used for conditional execution of the code.
- This tag is a container tag.
- It allows the execution of the body if the test attribute evaluates to true.
- **Syntax:**

```
<c:if test = "testCondition" var = "varName"
      scope = "page/request/session/application">
    Body Content
</c:if>
```

where,

- ❑ test specifies the test condition.
- ❑ var specifies the name of the variable of the test condition.
- ❑ scope specifies the scope of the variable, var.

- In <c:if> tag, attribute var and scope are optional.

© Aptech Limited. Web Component Development Using Java/Session 12 19

Decision-making Tags 3-5

- ❖ The code snippet demonstrates the use of if tag to evaluate.

```
<!-- Test the username -->
<c:if test="${name=='admin'}">
  <h4> Valid User </h4>
</c:if>
```

- ❖ The code uses if tag to check the value of the name variable.
- ❖ If the condition evaluates to true, that is, the value assigned to the name variable is admin, then the body gets executed.

© Aptech Limited. Web Component Development Using Java/Session 12 20

Decision-making Tags 4-5

<c:choose>

- The tag is similar to the switch statement in Java.
- The <c:choose> tag performs conditional block execution.
- The <c:choose> tag processes body of the <c:when> tag.
- Multiple <c:when> tags can be embedded in a <c:choose> tag.
- If none of the conditions evaluates to true, then the body of <c:otherwise> tag is processed.

Syntax:

```

<c:choose>
    <c:when test = "testcondition">
        Body Content
    </c:when>
    ...
    ...
    <c:otherwise>
        Body Content
    </c:otherwise>
</c:choose>

```

where,

- <c:when> is the body of <c:choose>. It will execute the body content if the test condition evaluates to true.
- <c:otherwise> is executed when none of the test conditions of <c:when> evaluates to true.

© Aptech Limited. Web Component Development Using Java/Session 12. 21

Decision-making Tags 5-5

- The code snippet demonstrates the choose tag.

```

// Performs the checking of the condition
<c:choose>
    <c:when test="${name == 'admin'}">
        <h4> Valid User </h4>
    </c:when>
    <c:otherwise>
        <h4> Invalid User </h4>
    </c:otherwise>
</c:choose>

```

- The choose tag performs evaluation of the condition to test if name entered is 'admin'.
- If the condition evaluates to true, then the nested when tag gets executed, else the nested otherwise tag gets executed.

© Aptech Limited. Web Component Development Using Java/Session 12. 22

Use slides 18 to 22 to explain decision-making tags.

Tell the students that <c:if> is used for testing conditions. <c:if> tag evaluates a condition and executes a block of code if the result is true. It is more or like an if statement in java which evaluates a condition and executes a block of code if the result is true.

The set of statements enclosed within <c:if> tag gets executed if test="true". For using this tag the developer generally use expression language to evaluate a relational expression. The developer can use EL because it returns boolean value(true/false) after evaluating the condition and we need the boolean value for testing attribute.

Then, explain the syntax of the <c:if> tag which is as follows:

```

<c:if test="${condition}">
...
...
</c:if>

```

Then, explain the code snippet demonstrating the use of the if tag as mentioned on slide 20. Following is the additional code snippet used checking the age of the person.

```

<c:set var="age" value="50"/>
<c:if test="${age >= 18}">
    <c:out value="You are eligible for voting!" />
</c:if>

<c:if test="${age < 18}">
    <c:out value="You are not eligible for voting!" />
</c:if>

```

The developer can also store the result of the `if` condition in a variable.

```

<c:if test="${17 >= 18}" var="res" scope="request">
</c:if>

```

To print the value of the variable “`res`” on the page, the `<c:out>` tag can be used.

```
<c:out value="${requestScope.res}" />
```

The `choose` tag performs conditional block execution by the embedded when subtags. It renders the body of the first when tag whose test condition evaluates to true. If none of the test conditions of nested when tags evaluates to true, then the body of an otherwise tag is evaluated, if present.

`<c:choose>`, `<c:when>`, and `<c:otherwise>` tags forms conditional logic. These tags are used to implement ‘if-else’ logic. These tags are also used to provide alternate execution.

Then, explain the syntax of using `<c:choose>` tag. Also, explain how to use `<c:choose>` tag.

Following is the additional code snippet to test the category to which members belongs using

`<c:choose> tag.`

```

<c:choose>
    <c:when test="${customer.category == 'trial'}" >
        ...
    </c:when>
    <c:when test="${customer.category == 'member'}" >
        ...
    </c:when>
    <c:when test="${customer.category == 'preferred'}" >
        ...
    </c:when>
    <c:otherwise>
        ...
    </c:otherwise>
</c:choose>

```

Slides 23 to 27

Let us understand iteration tags.

Iteration Tags 1-5

- ❖ The iteration tag is required for performing looping function.
- ❖ The object can be retrieved from a collection in the JavaBeans components and assigned to a scripting variable by using iteration tags.
- ❖ The two types of iteration tags are as follows:
 - ❑ <c:forEach>
 - ❑ <c:forTokens>

© Aptech Limited. Web Component Development Using Java/Session 12 23

Iteration Tags 2-5

<c:forEach>

- This tag is used to repeat the body content over a collection of objects.
- The iteration will continue for a number of times specified by the user in the code.
- **Syntax:**

```
<c:forEach var = "varName" item = "collection"
           varStatus = "varStatusName" begin = "begin" end =
           "end" step = "step">
    Body Content
</c:forEach>
```

where,

- ❑ var specifies the name of the exported scoped variable.
- ❑ item specifies the collection of items to iterate over.
- ❑ varStatus specifies the name of the variable for the status of iteration.
- ❑ begin specifies the index from which the iteration is to begin.
- ❑ end specifies the index at which the iteration is to end.
- ❑ step specifies that iteration will process every item of the collection.

© Aptech Limited. Web Component Development Using Java/Session 12 24

Iteration Tags 3-5

- ❖ The code snippet demonstrates the forEach tag.

```
// Displays objects from collection
"companies"
<c:forEach var= "company" items =
"${companies}"> ${company} <br>
</c:forEach>

// Displays values from 10 to 100
<c:forEach var="i" begin="10" end="100">
    ${i}
</c:forEach>
```

© Aptech Limited. Web Component Development Using Java/Session 12 25

Iteration Tags 4-5

<c:forTokens>

- It is used to iterate over a collection of tokens separated by user-specified delimiters.
- It is a container tag.
- Syntax:

```
<c:forTokens items="stringofToken" delims="delimiters" var="varName" varStatus="varStatusName" begin="begin" end="end" step="step">
    Body Content
</c:forTokens>
```

where,

- Items specifies the string of value to iterate.
- delims specifies the character that separates the tokens in the string.
- var specifies the name of the scope variable for the item of iteration.
- varStatus specifies the name of the scope variable for the status of iteration.

© Aptech Limited. Web Component Development Using Java/Session 12. 26

Iteration Tags 5-5

❖ The code snippet demonstrates the <c:forTokens> tag.

```
// Displays each token at a time separated by
// ',' delimiter
<c:forTokens var="token" items="Tom,Dick,Harry"
delims=",">

    <c:out value="${token}" />
</c:forTokens>
```

© Aptech Limited. Web Component Development Using Java/Session 12. 27

Use slides 23 to 27 to explain iteration tags. Tell the students that <c:forEach> tag executes the same set of statements for a specific number of times. <c:forEach> tag in JSTL is used for executing the same set of statements for a finite number of times. It allows you to iterate over a collection of objects.

Then, explain the syntax of using <c:forEach> tag on the JSP page which is as follows:

```
<c:forEach var="counter_variable_name" item="collection" varStatus="varStatusName" begin="intial_value" end="final_limit">
    //Block of statements
</c:forEach>
```

The “item” specifies the collection of items to iterate over, whereas the “var” specifies the name of the scoped variable.

Then, explain the code snippet on slide 25 to iterate through the collection.

Then, explain the <c:forTokens> tag which works with delimiter. <c:forTokens> tag is used to break the input data into multiple parts based on the delimiter.

Then, explain the syntax of `<c:forToken>` tag which is as follows:

```
<c:forTokens items = "stringofToken" delims = "delimiters" var =  
"varName" varStatus = "varStatusName" begin = "begin" end = "end"  
step = "step">  
    // Body Content  
</c:forTokens>
```

Here, `delims` specifies the character that separates the tokens in the string.

Then, explain the code snippet demonstrating the use of `<c:forToken>` tag mentioned on slide 27.

In-Class Question:

After you finish Iteration tags, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which tag is used to break the input data into multiple parts based on the delimiter?

Answer:

`<c:forTokens>` tag

Slide 28

Let us understand SQL tag library.

The slide has a blue header bar with the title 'SQL Tag Library'. Below the header, there is a bulleted list of three items:

- ❖ JSTL SQL tag library is used to interact with other databases such as Oracle, MySQL, or Microsoft SQL Server.
- ❖ These tags are prefixed by `sql`.
- ❖ SQL tags include:

Below the list, there is a code block showing five JSTL SQL tags:

```
<sql:setDataSource>
<sql:query>
<sql:update>
<sql:transaction>
<sql:param>
```

At the bottom of the slide, there are three small pieces of text: '© Aptech Limited.', 'Web Component Development Using Java/Session 12', and '28'.

Use slide 28 to explain SQL tag library. Tell students that the JSTL SQL tags are used for accessing databases. The JSTL SQL tag interacts with relational databases such as Oracle, MySQL, or Microsoft SQL Server. Using SQL tags we can run database queries. These tags are prefixed by `sql`.

Slides 29 to 31

Let us understand `<sql:setDataSource>` tag.

<sql:setDataSource> 1-3

- ❖ JSTL SQL tags are used to access databases.
- ❖ They are designed for low-volume Web-based applications.
- ❖ SQL tag library provides tags that allow direct database access within a JSP page.
- ❖ The JSTL SQL tag provides the following functionalities:

Passing Database Queries	• The functionality allows executing queries using the <code><sql:query></code> tag.
Accessing Query Results	• The functionality allows the users to access results for queries.
Database Modifications	• The functionality helps in modifying database using the <code><sql:update></code> tag.

© Aptech Limited. Web Component Development Using Java/Session 12 29

<sql:setDataSource> 2-3

- ❖ Syntax:

```
<sql:setDataSource dataSource = "datasource" | url
= "jdbcurl"
driver = "driverclassname"
user = "username" password = "userpwd"
var = "varname"
scope = "page/request/session/application"/>
```

where,

- ❑ `dataSource` can either be the path to Java Naming and Directory Interface (JNDI) resource or a JDBC parameter string.
- ❑ `url` is the URL associated with the database.
- ❑ `driver` is a JDBC parameter and takes the driver class name.
- ❑ `user` takes the user of the database.
- ❑ `password` takes the user password.
- ❑ `var` is the name of exported scoped variable for the data source specified.
- ❑ `scope` specifies the scope of the variable.

© Aptech Limited. Web Component Development Using Java/Session 12 30

<sql:setDataSource> 3-3

- ❖ The code snippet demonstrates the use of `<sql:setDataSource>` tag to make the connection to SQL Server database.

```
// Sets the data source for the database
<sql:setDataSource
driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://10.1.3.27:1433;DataBaseName=pubs;" user="sa" password="playware"
var="conn"/>
```

- ❖ In the code:

- ❑ `<sql:setDataSource>` is used to set a data source for the database.
- ❑ This is an empty tag and allows the user to set data source information for the database.
- ❑ The `url` attribute provides the JDBC url string for the SQL Server database.

© Aptech Limited. Web Component Development Using Java/Session 12 31

Use slides 29 to 31 to explain `<sql:setDataSource>` tag. Mention that the `<sql:setDataSource>` tag sets the data source configuration variable.

<sql:setDataSource> tag saves the data-source information in a scoped variable that can be used as input to the other JSTL database action. The <sql:setDataSource> tag creates data source variable directly from JSP and it can be stored in a scoped variable with the help of scoped attribute. It can be used as input to other database actions.

Attributes of <sql:setDataSource> tags are

- `datasource` which specifies data source name.
- `driver` which specifies JDBC driver class name used to create a connection.
- `url` which specifies JDBC URL associated with database.
- `user` which specifies database username.
- `password` which specifies database password.
- `var` which specifies name of the variable which is used to store created data source object.
- `scope` which specifies scope of the variable.

Then, explain the code snippet that demonstrates the use of <sql:setDataSource> sql tag to make the connection to SQL Server database.

Tell them they can also set the Java Naming and Directory Interface (JNDI) name to get the data source information. For example, <sql:setDataSource dataSource="jdbc/BookDB" />

Slides 32 to 34

Let us understand `<sql:query>` tag.

<sql:query> 1-3

- ❖ The `<sql:query>` tag searches the database and returns a result set containing rows of data.
- ❖ The tag can either be an empty tag or a container tag.
- ❖ The `SELECT` statement is used to select data from the table.
- ❖ Syntax:

```
<sql:query sql = "sqlQuery" var = "varName" scope
= "{page|request|session|application}" dataSource
= "dataSource" maxRows = "maxRows" startRow =
"startRow"/>
```

where:

- `sql` specifies the SQL query statement.
- `var` specifies the name of the exported scope variable for the query result.
- `scope` specifies the scope of the variable.
- `dataSource` specifies the data source associated with the database to query.
- `maxRows` specifies the maximum number of rows to be included in the result.
- `startRow` specifies the row starting at the specified index.

© Aptech Limited. Web Component Development Using Java/Session 12 32

<sql:query> 2-3

- ❖ Syntax:

```
<sql:query var = "varName" dataSource =
"dataSource" scope =
"{page|request|session|application}" maxRows =
"maxRows" startRow = "startRow">
    SQL Statement
    <sql:param/>
</sql:query>
```

where,

- `param` takes the parameter for the query.

© Aptech Limited. Web Component Development Using Java/Session 12 33

<sql:query> 3-3

- ❖ The code snippet demonstrates the use of `<sql:query>` tag.

```
// Gets the records about the 'product' from data
source 'conn'
<sql:query var="products" dataSource="${conn}">
    select * from Products
</sql:query>
```

© Aptech Limited. Web Component Development Using Java/Session 12 34

Use slides 32 to 34 to explain `<sql:query>` tag. Tell the students the `<sql:query>` tag is used to execute a query specified in `sql` attribute or in the body. It executes SQL statement and saves in the scoped variable.

Attributes of `<sql: query>` tag are:

- `sql` which specifies SQL command that has to be executed.

- **var** which is used to specify variable name used to store result of query.
- **scope** which specifies scope of the variable.
- **datasource** which specify data source name.
- **maxRows** which specifies maximum number of rows that has to be stored in the variable.
- **startRows** which specifies the starting row number.

Then, explain the code snippet to get the record from the database using the **<sql:query>** tag given on slide 24.

Following is the additional code snippet that runs the parameterized query:

```
<c:set var="bookID" value="${param.Add}" />
<sql:query var="books" >
    select * from books where id = ?
    <sql:param value="${bookID}" />
</sql:query>
```

Slides 35 to 37

Let us understand <sql:update> tag.

<sql:update> 1-3

- ❖ The <sql:update> tag executes the INSERT, UPDATE, and DELETE statements.
- ❖ It returns 0, if no rows are affected by the DML statements.
- ❖ Syntax:

```
<sql:update sql = "sqlUpdate" dataSource =
"dataSource" var = "varName" scope =
"(page|request|session|application)"/>
```

where,

- ❑ sql specifies the update, insert, or delete statement.
- ❑ dataSource specifies the data source associated with the database to update.
- ❑ var specifies the name of the exported scope variable for the result of the database update.
- ❑ scope specifies the scope of variable, such as page, request, session, or application.

© Aptech Limited. Web Component Development Using Java/Session 12 35

<sql:update> 2-3

- ❖ Syntax:

```
<sql:update dataSource = "dataSource" var =
"varName" scope =
"(page|request|session|application)">
    SQL Statement
    <sql:param value = "value"/>
</sql:update>
```

where,

- ❑ update is the UPDATE statement in SQL.
- ❑ param takes the parameter for the query.

© Aptech Limited. Web Component Development Using Java/Session 12 36

<sql:update> 3-3

- ❖ The code snippet demonstrates the use of <sql:update> tag.

```
// Adds the product details in the Product table
<sql:update var="newrow" dataSource="${conn}"
INSERT INTO Products(ProductName, ProductType, Price,
Brand, Description)
VALUES('Jeans', 'Clothes', '1000', 'Lee', 'Good Quality
Jeans')
</sql:update>
```

© Aptech Limited. Web Component Development Using Java/Session 12 37

Use slides 35 to 37 to explain `<sql:update>` tag. The `<sql: update>` tag executes SQL statement specified in the `sql` attribute. The `<sql:update>` tag executes the INSERT, UPDATE, and DELETE statements. It returns 0, if no rows are affected by the DML statements.

Attributes of `<sql:setupdate>` tag are:

- `sql` which specifies SQL command that has to be executed.
- `var` which specifies the variable name used to store result of query.
- `scope` which specifies scope of the variable.
- `datasource` which specify data source name.

Then, explain them the code snippet demonstrating the use of the `<sql:update>` tag.

Slides 38 and 39

Let us understand `<sql:transaction>` tag.

<sql:transaction> 1-2

- ❖ The `<sql:transaction>` is used to establish a transaction context for `<sql:query>` and `<sql:update>` tags.
- ❖ The connection object is obtained from `<sql:transaction>`.
- ❖ This tag is responsible for managing access to the database.
- ❖ **Syntax:**

```
<sql:transaction dataSource = "dataSource" isolation =
    isolationLevel>
        <sql:update> or <sql:query> statements
    </sql:transaction>
```

where,

- ❑ `dataSource` sets the SQL data source which can be a string or a data source object.
- ❑ `isolation` sets the transaction isolation level. Isolation level can be `read_committed`, `read_uncommitted`, `repeatable_read`, or `Serializable`.

© Aptech Limited. Web Component Development Using Java/Session 12 38

<sql:transaction> 2-2

- ❖ The code snippet demonstrates the use of `<sql:transaction>` tag.

```
/** The code snippet performs transaction by first accessing a
data source to create a table and then inserting a row. It then
performs a SQL queries on the table
*/
<sql:transaction dataSource="${mydatasource}">
    <sql:update var="newTable">
        create table emp (
            id int primary key,
            name varchar(80)
        )
    </sql:update>
    <sql:update var="updateCount">
        INSERT INTO emp VALUES (1,'Jenny')
    </sql:update>
    <sql:update var="updateCount">
        INSERT INTO emp VALUES (2,'Christina')
    </sql:update>
    ...
    <sql:query var="empQuery">
        SELECT * FROM emp
    </sql:query>
</sql:transaction>
```

© Aptech Limited. Web Component Development Using Java/Session 12 39

Use slides 38 and 39 to explain `<sql:transaction>` tag. It provides connection to all database operations and executes all statements in one transaction. JSTL Transaction tag provides the capability to run SQL statement in a group. User can run multiple SQL statements at a time. It is called one single transaction. In this transaction either all statements run successfully or all fail if one statement does not run successfully.

Attributes of `<sql:transaction>` tag are:

- `dataSource` which specifies datasource for the connection.
- `isolation` which specifies level of transaction isolation that can be **can be** `read_committed`, `read_uncommitted`, `repeatable_read`, or `Serializable`.

The, explain the code snippet demonstrating the use of `<sql:transaction>` tag given on slide 39.

Slides 40 to 42

Let us understand `<sql:param>` tag.

<sql:param> 1-3

- ❖ `<sql:param>` is used to set values for parameters markers ('?') in SQL statements.
- ❖ It acts as a sub tag for `<sql:query>` and `<sql:update>`.
- ❖ **Syntax:**

```
<sql:param value = "value"/>
```

where,

- `value` sets the value for the parameter.

© Aptech Limited. Web Component Development Using Java/Session 12 40

<sql:param> 2-3

- ❖ The code snippet demonstrates how to use the `<sql:param>` in database updation.

```
//The code snippet sets the values for the
parameters//  
<sql:update>
    INSERT INTO Employee (DemoFirstName,
    DemoLastName, EmpDate)
    VALUES(?, ?, ?)
    <sql:param value="${param. DemofirstName}" />
    <sql:param value="${param. DemolastName}" />
    <sql:dateParam value="${empDemoDate}"
    type="date" />
</sql:update>
```

© Aptech Limited. Web Component Development Using Java/Session 12 41

<sql:param> 3-3

- ❖ Following figure depicts the use of `<sql:param>` tag.

```
<sql:query var="product"
    sql="select * from PUBLIC.product where id = ?" >
    <sql:param value="#{productId}" />
</sql:query>

<c:forEach var="productRow" begin="0"
    items="#{product.rows}">
    <sql:update var="product" sql="update PUBLIC. product set
        inventory = inventory - ? where id = ?" >
        <sql:param value="#{item.quantity}" />
        <sql:param value="#{productId}" />
    </sql:update>
</c:forEach>
```

© Aptech Limited. Web Component Development Using Java/Session 12 42

Use slides 40 to 42 explain `<sql:param>` tag. Tell the students that JSTL `param` tag provides the value of the parameter. Parameters are sent using `param` tag in an SQL update statement.

<sql:param> tag sets parameter values for SQL statement. This tag provides value for value placeholder by using actions of both <sql:query> and <sql:update>.

Then, explain the code snippet provided in the figure on slide 42 for using the <sql:param> tag.

Slide 43

Let us summarize the session.

Summary

- ❖ JSTL provides a set of reusable standard tags.
- ❖ JSTL standard tag library works in a similar manner everywhere and this makes the iteration over the collection using scriptlets unnecessary.
- ❖ JSTL allows programming using tags rather than scriptlet code.
- ❖ The core tag library has general purpose tags that are used to manipulate scoped variables created within a JSP page.
- ❖ Decision-making tags are used to do conditional processing of code in a JSP page.
- ❖ Iteration tags are used to iterate over a collection of objects multiple times.
- ❖ SQL Tag Library is useful in performing database queries. It allows easy access to query results.
- ❖ The database statements, such as insert, update, and delete can be performed by SQL tags.

© Aptech Limited. Web Component Development Using Java/Session 12 43

In slide 43, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

12.3 Post-Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also explore the custom tags in JSP.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 13 – JSP Custom Tags

13.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

13.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain JSP custom tags in JSP
- Describe the common terminology used in JSP custom tags
- Explain the working of custom tag libraries
- Explain the different types of custom tags available in JSP
- Explain how to create classic custom tags
- Explain use of Tag Extension API
- Explain Simple Tags API

13.1.2 Teaching Skills

To teach this session successfully, you should be aware with common terminology related to JSP Custom tags used in JSP. Familiarize yourself with working of custom tag libraries and the different types of custom tags available in JSP.

Aware yourself with how to create classic custom tags, the use of Tag Extension API and also Simple Tags API.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❖ Explain JSP custom tags in JSP
- ❖ Describe the common terminology used in JSP custom tags
- ❖ Explain the working of custom tag libraries
- ❖ Explain the different types of custom tags available in JSP
- ❖ Explain how to create classic custom tags
- ❖ Explain use of Tag Extension API
- ❖ Explain Simple Tags API

© Aptech Ltd. Web Component Development Using Java/Session 13 2

Tell the students that they will be introduced to Custom tags and their common terminology used in JSP custom tags. Explain the working of custom tag libraries and the different types of custom tags available in JSP.

Finally, explain how to create classic custom tags, the use of Tag Extension API, and also Simple Tags API.

13.2 In-Class Explanations

Slide 3

Let us understand JSP Custom tags.

JSP Custom Tags

- ❖ Custom tag allows Java developers to embed Java code in JSP pages.
- ❖ Using custom tags in a JSP page involves three steps:

Creating a Tag Library Descriptor (TLD)

- TLD file contains the information on each tag available in the library.
- It is an XML document.
- It used to validate the tags.

Creating a tag handler class

- Tag handler is a Java class that defines a tag described in the TLD file.

Creating a JSP page that will access the custom tags

- JSP page will use the tags defined in a tag library by using a taglib directive in the page before any custom tag is used.

© Aptech Ltd. Web Component Development Using Java/Session 13 3

Use slide 3 to explain JSP Custom tags.

Tell the students that JSP tags simplifies the development and maintenance of JSP pages. Custom tags performs following functions such as operating on implicit objects, processing forms, accessing databases, and other enterprise services such as email and directories, and implementing flow control.

Custom tags increase productivity as they can be reused in more than one Web application.

Custom tags are distributed as a tag library. Tag handler is an object that implements a custom tag. Custom tags are user-defined tags.

Advantages of Custom Tags are as follows:

- Eliminates the need of scriptlet tag
- Separates business logic from JSP
- Reusability

In-Class Question:

After you finish explaining JSP Custom tags, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which is the one of the main advantage of using tags for non-programmers?

Answer:

Eliminates the need of scriptlet tag

Slides 4 to 7

Let us understand Custom tags terminology.

Custom Tags Terminology 1-4

- ❖ Tag library descriptor
 - Is a XML file, which describes custom tags in a JSP program.
 - Contains the definitions of a custom tag and is saved with an extension.tld.
 - Imported to the JSP program by using the <%@taglib> directive.
- ❖ The JSP container creates an instance of the imported library descriptor file and traces the handler class of the custom tag.

© Aptech Ltd. Web Component Development Using Java/Session 13 4

Custom Tags Terminology 2-4

- ❖ The code snippet shows a sample tag library descriptor that describes a tag called Name.

```
<taglib version="2.0" xmlns="http://java.sun.com/xml/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-jsp>taglibrary_2_0.xsd">
    <!-- The tag library and the tag is described inside this tag-->
    <taglib>
        <!-- The version of the tag library -->
        <tlib-version>2.0</tlib-version>
        <!-- The JSP specification version for the tag library -->
        <jsp-version>2.0</jsp-version>
        <!-- This is the name assigned to the tag library -->
        <short-name>tags</short-name>
        <!-- This specifies the path of the TLD file-->
        <uri>tag lib version id</uri>
    </taglib>

```

© Aptech Ltd. Web Component Development Using Java/Session 13 5

Custom Tags Terminology 3-4

```
<!-- Provides a short description of the tag library-->
<description>The tag library </description>
<!-- Provides a detailed description of the tag-->
<tag>
    <!-- This is the name of the tag used in the JSP page-->
    <name>name</name>
    <!-- This is the name of the tag handler class for the concerned tag-->
    <tag-class>tags.NameTag</tag-class>
    <!-- This specifies the type of body content. It can be empty, JSP or tag-dependent -->
    <body-content>empty</body-content>
    <!-- This describes the attribute passed with the tag-->
    <attribute>
        <!-- This specifies the name of the attribute passed with the tag-->
        <name>name</name>
        </attribute>
    </tag>
</taglib>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 6

Custom Tags Terminology 4-4

❖ **Tag handler:**

- It is a simple Java class file that contains the code for the functionality of the custom tag in the JSP program.
- These are of two types, classic and simple.
- Figure depicts the custom tags terminology.

```
Tag library descriptor
<taglib>
  body
    <tag>
      body
        <attribute>
          Description
        </attribute>
      </tag>
    </taglib>
```

Tag library descriptor



Tag handler class{}

© Aptech Ltd. Web Component Development Using Java/Session 13 7

Use slides 4 to 7 to explain custom tags terminology.

Tell the students that there are two types of components for a tag library. They are as follows:

- Tag library descriptor file
- Tag handler

A Tag Library Descriptor (TLD) file is an XML document that describes the library and contains information about each tag contained in the library. TLDs are used to validate the tags.

Using slides 5 and 6 show them a sample TLD file.

Then, using slide 7, tell them that you write a custom JSP tag by writing a Java class called a tag handler.

Slides 8 and 9

Let us understand tag libraries.

Tag Libraries 1-2

- ❖ It is a collection of custom tags.
- ❖ It allows the developer to write reusable code fragments, which assign functionality to tags.
- ❖ When a tag is used in a JSP page, the library containing the tag has to be imported into the JSP page using the `<%@taglib>` directive.
- ❖ **Syntax:**

```
<%@taglib prefix="u" uri="/WEB-INF/tlds/sampleLib.tld" %>
```

where,

- `uri` is the path that uniquely identifies the TLD file of the tag
- `prefix` is the prefix attribute that distinguishes various tags in the JSP page

© Aptech Ltd. Web Component Development Using Java/Session 13 8

Tag Libraries 2-2

- ❖ Figure depicts the working of custom tag libraries.

The diagram shows the flow of information from a JSP page to a Tag Library. A JSP page contains a tag directive (`<%@taglib...>`) which points to an Application Descriptor (web.xml). This descriptor points to a Tag Library Descriptor (*.tld) and a Tag Library. The Tag Library contains Taghandler1.class, Taghandler2.class, Taghandler3.class, and Taghandler4.class.

© Aptech Ltd. Web Component Development Using Java/Session 13 9

Use slides 8 and 9 to explain tag libraries.

Tell the students that tag library defines a set of related custom tags and contains the objects that implement the tags.

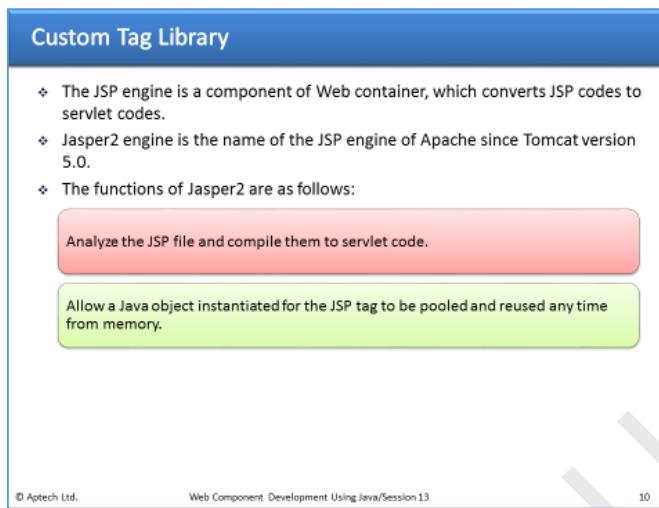
Tag library is a collection of custom tags. This library allows the developer to write reusable code fragments, which assign functionality to tags. The reference to each tag is stored in the tag library. When a tag is used in a JSP page, the library containing the tag has to be imported into the JSP page using the `<%@taglib>` directive.

The syntax of the tag directive to be included on the JSP page is as follows: `<%@taglib prefix="u" uri="/WEB-INF/tlds/sampleLib.tld" %>`

Using slide 9, explain the process of accessing the tag library on the JSP page with the help of the figure as shown on slide 9.

Slide 10

Let us understand role of JSP engine in analysing the custom tag library.



The slide has a blue header bar with the title "Custom Tag Library". Below the header is a bulleted list of three points:

- ❖ The JSP engine is a component of Web container, which converts JSP codes to servlet codes.
- ❖ Jasper2 engine is the name of the JSP engine of Apache since Tomcat version 5.0.
- ❖ The functions of Jasper2 are as follows:

Below the list are two callout boxes:

- Analyze the JSP file and compile them to servlet code.
- Allow a Java object instantiated for the JSP tag to be pooled and reused any time from memory.

At the bottom of the slide, there is footer text: © Aptech Ltd., Web Component Development Using Java/Session 13, and the number 10.

Use slide 10 to explain the role of JSP engine in analysing the custom tag library.

Tell them that the Jasper engine converts JSP code to servlet code. The functionality of Jaspers 2 is as follows:

- Analyze the JSP file and compile them to servlet code.
- Allow a Java object instantiated for the JSP tag to be pooled and reused any time from memory.

Tips:

JSP tag libraries include one or more custom JSP tags and are defined in a tag library descriptor (.tld) file.

Slide 11

Let us understand locating TLD file.

Locating TLD File

- ❖ TLD file can reside in either any directory of a Web application such as:
 -/WEB-INF/sampleLib.tld
 -/WEB-INF/tld/sampleLib.tld
- ❖ They can be package in a jar file and place that jar in/WEB-INF/lib/ directory along with related jars and any related resources.

© Aptech Ltd. Web Component Development Using Java/Session 13 11

Use slide 11 to explain how to locate TLD file.

Tell them that TLD file resides in the WEB-INF folder of the war archive file. It can either be located in any of these directories:

-/WEB-INF/sampleLib.tld
-/WEB-INF/tld/sampleLib.tld

Also, they can alternatively package the TLD library in a jar file and place that jar in/WEB-INF/lib/ directory along with related jars and any related resources.

Slides 12 to 15

Let us understand associating URIs with TLD file locations.

Associating URIs with TLD File Locations

- ❖ A specific URI for each .tld file refers to its tag in a JSP page.
- ❖ This is done by assigning path to the .tld file to uri attribute.
- ❖ The uri attribute is mapped to the .tld file in two ways namely, implicit and explicit mapping.
- ❖ In implicit mapping the container reads all the .tld files present in the jar.
- ❖ Then, the JSP container automatically creates a mapping between the URI specifying the JAR location and the TLD file present inside the JAR file.
- ❖ **Syntax:**

```
<%@ taglib prefix="u" uri="/WEB-INF/TagJARfile.jar" %>
```

© Aptech Ltd.

Web Component Development Using Java/Session 13

12

Explicit Mapping

- ❖ Is a method to map the URI to a TLD file.
- ❖ Defines reference to the tag library descriptor by the <taglib> element in the deployment descriptor, web.xml, of the Web application.
- ❖ Each <taglib> element contains two sub elements as follows:

<taglib-uri>	• This is the URI identifying a tag library.
<taglib-location>	• It describes the path to the tag library descriptor file for the custom tag.

- ❖ The code snippet uses the tag <taglib-location> element to indicate relative path of AttributeTag.tld.

```
<taglib>
  <taglib-uri>AttributeTag</taglib-uri>
  <taglib-location>/WEB-INF/tlds/AttributeTag.tld</taglib-location>
</taglib>
```

© Aptech Ltd.

Web Component Development Using Java/Session 13

13

Resolving URIs to TLD File Locations

- ❖ In explicit mapping, the association of the URIs to the TLD files is called the resolution of URIs.
- ❖ This association is necessary to locate the tag library descriptor file, which defines the tag.
- ❖ If the value of the uri attribute matches any of the <taglib-uri> entries, the engine uses the value of the corresponding <taglib-location> to locate the actual TLD file.

© Aptech Ltd.

Web Component Development Using Java/Session 13

14

Tag Library Prefix

- ❖ To distinguish custom tags from one another, they are named uniquely by using prefix attributes in the `<%@taglib>` directive.
- ❖ The prefix of a tag is also used as a reference by the container to invoke its respective TLD file and tag handler class.

❖ **Syntax:**

```
<%@taglib prefix="ui" uri="/WEB-INF/tlds/sampleLib_ui.tld"
%>
<%@taglib prefix="logic" uri="/WEB-INF/tlds/sampleLib_logic.
tld" %>
<%@taglib prefix="data" uri="/WEB-
INF/tlds/sampleLib_data.tld" %>
```

Use slides 12 to 15 to explain how to associate URIs with TLD file locations.

Tell them as discussed in the syntax of the tag directive in JSP, the `uri` attribute specifies the URI of the .tld file. The URI specifies the location of the tag library. The JSP engine attempts to find the tag library descriptor by matching the `uri` attribute to the `uri` that is defined in the Web application deployment descriptor (`web.xml`) with the `<taglib-uri>` element.

This is basically done by assigning path to the .tld file to `uri` attribute. The `uri` attribute is mapped to the .tld file in two ways namely, implicit and explicit mapping.

In implicit mapping the container reads all the .tld files present in the jar, then the JSP container automatically creates a mapping between the URI specifying the JAR location and the TLD file present inside the JAR file.

For example, `<%@ taglib prefix= "u" uri="/WEB-INF/TagJARfile.jar" %>`

Use slide 13 to explain explicit mapping. The explicit mapping is another method to map the URI to a TLD file. This is done by defining reference to the tag library descriptor by the `<taglib>` element in the deployment descriptor, `web.xml`, of the Web application.

For example, consider a `<%@taglib>` directive specified in the JSP page as: `<%@taglib uri='myTLD' prefix='mytaglib'%>`

Here, `myTLD` specifies the name of the taglib-uri mentioned in the `web.xml` file. The `myTLD` refers to the `mytaglibrary.tld` file.

The `web.xml` file will be as follows:

```
<taglib>
  <taglib-uri>myTLD</taglib-uri>
  <taglib-location>mytaglibrary.tld</taglib-location>
</taglib>
```

Then, explain the `prefix` attribute which assigns a label to the tag library to be referred in the JSP page. For example, to access the custom tags defined in `mytaglibrary.tld` file which is named as `myTLD` in the `web.xml`, you can prefix it as `mytaglib` to use the tag in your JSP page.

In-Class Question:

After you finish explaining how to associate URIs with TLD file locations, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Name the two ways by which the `uri` attribute is mapped to the `.tld` file.

Answer:

Implicit and Explicit mapping

Slides 16 to 19

Let us understand using custom tags in the JSP pages.

Using Custom Tags in JSP Pages

- ❖ Following are the types of custom tags:

Custom Tags			
Empty tag	Tag with attribute	Custom tags with JSP code	Nested custom tags

© Aptech Ltd. Web Component Development Using Java/Session 13 16

Tags with Attributes

- ❖ The custom tags with the attributes are called as **parameterized tags**.
- ❖ Attributes are passed to the tags as arguments are passed to a method.
- ❖ The code snippet customizes the behavior of a custom tag.

```

<html>
<body>
<%@ taglib prefix="tagPrefix" uri="sampleLib.tld" %>
<h1>
<tagPrefix:TagsWithAttributesuserName="userName1"></h1>
</body>
</html>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 17

Custom Tags with JSP Code

- ❖ Custom tags can contain JSP code as content inside them.
- ❖ The JSP code is simply evaluated or manipulated before being evaluated.
- ❖ The code snippet shows that the manipulation can be of converting the body content to a string and then formatting it as per requirements.

```

<html><body>
<%@ taglib prefix="tagPrefix" uri="sampleLib.tld" %>
<tagPrefix:if condition="true">
UserName is: <%= request.getParameter("userName") %>
</tagPrefix:if>
</body></html>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 18

Nested Custom Tags

- ❖ These are tags declared inside the tags.
- ❖ This is analogous to the `<table>` tag in an html page, which has `<tr>` and `<td>` tags inside it.
- ❖ One custom tag can make a container for another custom tag.
- ❖ The container is called the parent of the tags inside it.
- ❖ The code snippet shows the nested custom tags.

```
<html><body>
<%@ taglib prefix="tagPrefix" uri="sampleLib.tld" %>
<tagPrefix:switch conditionValue='<%= request.
getParameter("userName")%>'>
    <tagPrefix:case caseValue=" userName1">
        First User
    </tagPrefix:case>
    <tagPrefix:case caseValue=" userName2" >
        Second User
    </tagPrefix:case>
</tagPrefix:switch>
</body></html>
```

© Aptech Ltd.

Web Component Development Using Java/Session 13

19

Use slides 16 to 19 to explain using custom tags in JSP pages.

Tell them that a custom tag format can be:

- An empty format referred to as empty tag. For example, `<mytaglib:newtag attr1="xx" attr2="yy" ... />`
 - A tag with body referred to as body tag.
- For example,

```
<mytaglib:newtag attr1="xx" attr2="yy" ... >
    // body of the tag
</mytaglib:newtag>
```

Notice, in both the case, they accept attributes that are passed to the tags as arguments are passed to a method.

Also, show the example provided on slide 17, to explain passing attributes to the custom tags on the JSP page.

Then, show the code snippet as given on slide 18, to explain the use of body tag.

Then using slide 19, explain the nested tag format. Nested custom tags are tags declared inside the tags. They are similar to some html tags with nested elements in them. For example, this is analogous to the `<table>` tag in an html page, which has `<tr>` and `<td>` tags inside it. Similarly, one custom tag can make a container for another custom tag. The container is called the parent of the tags inside it.

Then, explain the code snippet as given on slide 19, to explain the nested tags.

Slide 20

Let us understand Classic Custom tags.

Classic Custom Tags

- ❖ The classic custom tags help in creating customized tags for JSP pages.
- ❖ It uses a tag handler class, which implements Tag, IterationTag, and BodyTag interfaces or extends the classes TagSupport or BodyTagSupport.
- ❖ It is then described in the Tag Library Descriptor (TLD) file.

© Aptech Ltd. Web Component Development Using Java/Session 13 20

Use slide 20 to explain Classic Custom tags.

Tell them that you can create tag handler for the custom tags. The tag handler can be of two types:

- Classic Tag Handler
- Simple Tag Handler

Tell the students that Classic tag handlers must be used if scripting elements are required.

The classic custom tags were introduced in JSP 1.1 as an extension to the existing standard tags.

These tags help in creating customized tags for JSP pages.

The classic tag handler contains three interfaces namely, Tag, BodyTag, and IterationTag.

Slides 21 to 24

Let us understand <taglib> element.

<taglib> Element 1-2

- ❖ The <taglib> element is the top-level or root element of the tag library descriptor.
- ❖ This tag contains some sub tags inside it. They are as follows:

<tlib-version>	• Describes the version of the tag library implementation
<jsp-version>	• Defines the JSP version
<short-name>	• Uniquely identifies the tag library from the JSP page
<uri>	• Used as a unique resource identifier for the location of a tag library
<tag>	• Provides all the information required for a particular classic custom tag used in a JSP page

© Aptech Ltd.

Web Component Development Using Java/Session 13

21

<taglib> Element 2-2

- ❖ The code snippet illustrates the use of the <taglib> elements along with its sub elements.

```
<taglib version="2.0"
 xmlns="http://java.sun.com/xml/ns/j2ee"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-
 instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2e
 e web-jsptaglibrary_2_0.xsd">

<tlib-version>1.0</tlib-version>
<jsp-version>1.2</jsp-version>
<short-name>attributetag</short-name>
<uri>/WEB-INF/tlds/AttributeTag</uri>

</taglib>
```

© Aptech Ltd.

Web Component Development Using Java/Session 13

22

<tag> Element 1-2

- ❖ The <tag> element contains the following sub elements to describe the tag characteristics:

<name>	• Assigns a unique name to the classic custom tag inside a JSP page.
<tag-class>	• Contains the name of the tag handler class that describes the functionality of a particular classic custom tag.
<body-content>	• Contains the content type for the body of the tag <ul style="list-style-type: none"> • This tag can be passed values such as empty, JSP, and Body-Content.
<attribute>	• Describes the attribute passed in the tag handler class to the JSP page containing the classic custom tag.

© Aptech Ltd.

Web Component Development Using Java/Session 13

23

<tag> Element 2-2

The code snippet illustrates the use of the <tag> elements along with its sub elements.

```
<tag>
<name>AttributeTag</name>
<tag-class>AttributeTag</tag-class>
<body-content>empty</body-content>
</tag>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 24

Use slides 21 to 24 to explain <taglib> element.

Tell them that before learning how to create a handler, let us learn how to define the tag handler class in the .tld file.

Tell the students that the <taglib> element is used to define a tag library that defines the tags. Some of the elements of <taglib> are as follows:

- <tlib-version>
- <jsp-version>
- <short-name>
- <uri>
- <tag>

Then, discuss how to define the AttributeTag discussed earlier in the session. Using slide 22, show the mapping of the tag.

Use slides 23 and 24 to explain <tag> element. Tell the students that the <tag> element defines the tag class and its content type.

Explain the code snippet as shown on slide 24 to illustrate the <tag> elements along with its sub elements for defining the tag handler class, AttributeTag.

In-Class Question:

After you finish explaining <tag> element, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which tag contains the name of the tag handler class that describes the functionality of a particular classic custom tag?

Answer:

<tag-class>

Slides 25 to 28

Let us understand <attribute> element.

<attribute> Element 1-2

- ❖ The <attribute> element inside the <tag> element describes the attribute passed in the tag handler class to the JSP page through the classic custom tag.
- ❖ It contains five sub tags:
 - name:** The name of the attribute.
 - type:** The data type of the attribute.
 - required:** This element specifies whether the attribute is required or not.
 - rtpexprvalue:** This element specifies whether the attribute is evaluated dynamically or not.
 - description:** A brief description of the attribute.

© Aptech Ltd. Web Component Development Using Java/Session 13 25

<attribute> Element 2-2

- ❖ The code snippet demonstrates the attribute element.

```
<tag>
<name>MyTag</name>
<tag-class>pkg.MyTag</tag-class>
<body-content>JSP</body-content>

<attribute>
<name>attr1</name>
<required>true</required>
</attribute>
<attribute>
<name>attr2</name>
<required>false</required>
<rtpexprvalue>true</rtpexprvalue>
</attribute>
</tag>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 26

<body-content> Element 1-2

- ❖ It is defined within the <tag> element.
- ❖ It indicates whether the classic custom tag contains any content inside it or not.
- ❖ The body content can have one of three values as follows: **empty**, **JSP**, or **tagdependent**.

empty

- ❖ The code snippet shows the tag that does not contain any body inside it to process, then it is called an empty tag.

```
<tag>
<name>AttributeTag</name>
<tag-class>AttributeTag</tag-class>
<body-content>empty</body-content>
<attribute>
</attribute>
</tag>
```

© Aptech Ltd.

Web Component Development Using Java/Session13

27

<body-content> Element 2-2

JSP

- ❖ The code snippet shows the JSP body content includes any other custom tag, core tag, scripting elements, or html text inside it.

```
<tag>
<name>UpperCase</name>
<tag-class>Icase.UpperCase</tag-class>
<body-content>JSP</body-content>
</tag>
```

tagdependent

- ❖ The code snippet specifies that the tag has a body, but its content is not to be interpreted by the JSP engine.

```
<tag>
<name>Query</name>
<tag-class>Table.Query</tag-class>
<body-content>tagdependent</body-content>
</tag>
```

© Aptech Ltd.

Web Component Development Using Java/Session13

28

Use slides 25 to 28 to explain <attribute> and <body-content> element.

Tell them that if a tag has attributes associated with it, then each attribute must be described within the <tag> element. If an attribute is required by a tag, <required> is set to 'true' or 'yes'. To allow a runtime expression value to be used by the tag, the <rtepxvalue> is set to 'true' or 'yes'. For each attribute of a tag, a Bean-like getter/setter method needs to be defined in the handler class.

Tell them that the <body-content> element indicates whether the classic custom tag contains any content inside it or not.

For example, if the tag that does not contain any body inside it to process, then it will be marked as **EMPTY**. Similarly, if the JSP body content includes any other custom tag, core tag, scripting elements, or html text inside it, then it is marked as **JSP**. Also, if the tag has a body, but its content is not to be interpreted by the JSP engine, then it is specified as **Tagdependent**.

Use slides 27 and 28 to explain <body-content> element. Tell the students the <body-content> element is used to validate that a tag invocation has the correct body syntax and is used

by page-composition tools to assist the page author in providing a valid tag body. There are three possible values in <body-content> element.

- tagdependent: The body of the tag is interpreted by the tag implementation itself.
- empty: The body must be empty.
- scriptless: The body accepts only static text, EL expressions, and custom tags.

Slides 29 to 31

Let us understand Tag Extension API.

Tag Extension API

- ❖ It adds the tag functionalities to the language.
- ❖ It is part of the `javax.servlet.jsp.tagext` package.
- ❖ This contains a number of interfaces and classes, which have their exclusive methods.

© Aptech Ltd. Web Component Development Using Java/Session 13 29

Interfaces

- ❖ The three important interfaces defined in the tag extension API are as follows:

Tag
<ul style="list-style-type: none"> • It allows communication between a tag handler and the servlet of a JSP page. • It is useful when a classic custom tag is without any body or even if it has body, then it is not for manipulation.
IterationTag
<ul style="list-style-type: none"> • It is used by tag handlers that require executing the tag, without manipulating its content. • It extends to the Tag interface.
BodyTag
<ul style="list-style-type: none"> • It extends IterationTag and adds two methods for supporting the buffering of body contents: <code>doInitBody()</code> and <code>setBodyContent()</code>.

© Aptech Ltd. Web Component Development Using Java/Session 13 30

Classes

- ❖ The `tagext` package includes two important classes `TagSupport` and `BodyTagSupport` for implementing tag functionalities in a JSP page.

TagSupport
<ul style="list-style-type: none"> • It acts as the base class for most tag handlers which supports empty tag, tag with attributes, and a tag with body iterations. • It implements the Tag and IterationTag interfaces. • It contains methods such as <code>doStartTag()</code>, <code>doEndTag()</code>, and <code>doAfterBody()</code>.
BodyTagSupport
<ul style="list-style-type: none"> • It supports tags that need to access and manipulate the body content of a tag. • It implements the BodyTag interface and extends the TagSupport class. • It contains the following methods: <code>setBodyContent()</code> and <code>doInitBody()</code>.

© Aptech Ltd. Web Component Development Using Java/Session 13 31

Use slides 29 to 31 to explain Tag Extension API.

Use slide 30 to explain tag Interfaces used for developing classic tags.

Explain them that the three interfaces are as follows:

- **Tag** — Implements the `javax.servlet.jsp.tagext.Tag` interface if you are creating a custom tag that does not need access to its interface. The API also provides a convenience class `TagSupport` that implements the `Tag` interface and provides default empty methods for the methods defined in the interface.

Tell them that next slides explain the methods of `Tag` interface.

- **BodyTag** — Implements the `javax.servlet.jsp.tagext.BodyTag` interface if your custom tag needs to use a body. The API also provides a convenience class `BodyTagSupport` that implements the `BodyTag` interface and provides default empty methods for the methods defined in the interface. The `BodyTag` extends `Tag` which is a super set of the interface methods.
- **IterationTag** — Implements the `javax.servlet.jsp.tagext.IterationTag` interface to extend `Tag` by defining an additional method `doAfterBody()` that controls the reevaluation of the body.

Use slide 31 to explain tag handler classes.

Tell the students `TagSupport` class is used as the base class for new tag handlers. The `TagSupport` class implements the `Tag` and `IterationTag` interfaces and adds additional convenience methods including getter methods for the properties in `Tag`. `TagSupport` has one static method to facilitate coordination among cooperating tags.

The `BodyTagSupport` class implements the `BodyTag` interface and adds additional convenience methods.

Slide 32

Let us understand Exceptions.

The slide has a blue header bar with the title 'Exceptions'. Below it is a bulleted list: 'The tag handler classes use the exception classes which are defined in the javax.servlet.jsp package.' This is followed by two sections: 'JspException' (purple background) and 'JspTagException' (blue background). Each section contains a bulleted list of descriptions.

Exception Type	Description
JspException	It is thrown by a tag handler to indicate some unrecoverable error while processing a JSP page.
JspTagException	It is a sub class of the JspException, which is thrown when an error is encountered inside the tag handler class while processing any tag.

© Aptech Ltd. Web Component Development Using Java/Session 13 32

Use slides 32 to explain Exceptions.

Tell the students that an exception can be used by a tag handler class to indicate some unrecoverable error. This will result in an error page. These exceptions are defined in the javax.servlet.jsp package.

Slides 33 to 35

Let us understand the methods of Tag Interface.

Methods of Tag Interface 1-3

- ❖ **doStartTag()**
 - The tag handler invokes this method when a request is made to a tag or when the starting tag of the element is encountered.
 - It returns either of the two field constants, returns the SKIP_BODY constant if there is no body to evaluate or returns the EVAL_BODY_INCLUDE constant.
 - **Syntax:**

```
public int doStartTag() throws JspException
```
- ❖ **setPageContext()**
 - This sets the current page context.
 - This is called by the page implementation prior to doStartTag().
 - **Syntax:**

```
public void setPageContext(PageContext pc)
```

© Aptech Ltd. Web Component Development Using Java/Session 13 33

Methods of Tag Interface 2-3

- ❖ **doEndTag()**
 - The tag handler invokes this method when the JSP page encounters the end tag.
 - It generally follows the execution of the doStartTag() if the tag is empty.
 - This method returns the SKIP_PAGE constant if there is no need to evaluate the rest of the page.
 - It returns EVAL_PAGE constant if the rest of the page needs to be evaluated.
 - **Syntax:**

```
public int doEndTag() throws JspException
```
- ❖ **release()**
 - The container calls this method on the handler class when the tag handler object is no longer required.
 - **Syntax:**

```
public void release()
```

© Aptech Ltd. Web Component Development Using Java/Session 13 34

Methods of Tag Interface 3-3

- ❖ The code snippet demonstrates the methods of the Tag interface.

```
//The evaluation of start tag starts
public int doStartTag() throws JspException {
    try {
        pageContext.getOut().print("Creating Custom Tag by
            Implementing Tag Interface");
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    // The SKIP BODY is returned
    return SKIP_BODY;
}
//The evaluation of end tag starts
public int doEndTag() throws JspException {
    // the SKIP PAGE is returned
    return SKIP_PAGE;
}
//All the resources held by the tag handler is released
public void release() {
}
//The current value of the pageContext is set
public void setPageContext(PageContext pc) {
    this.pageContext = pc;
}
//All the resources held by the tag handler is released
public void release() {
}
```

© Aptech Ltd. Web Component Development Using Java/Session 13 35

Use slides 33 to 35 to explain the methods of Tag interface.

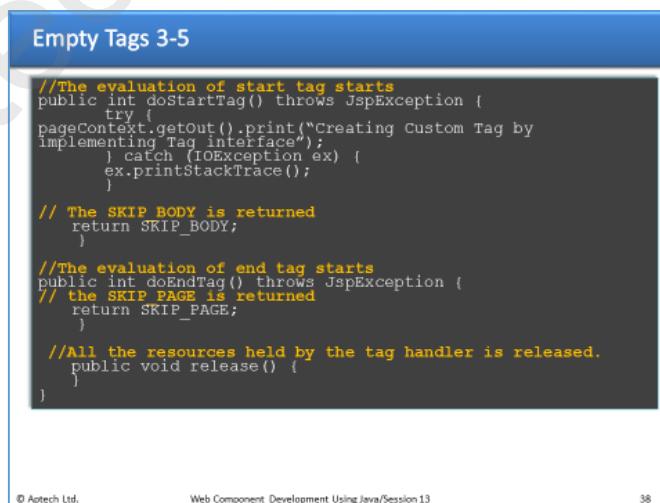
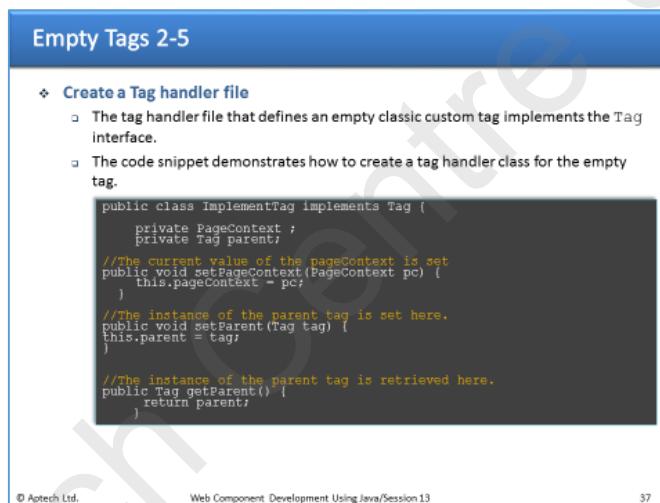
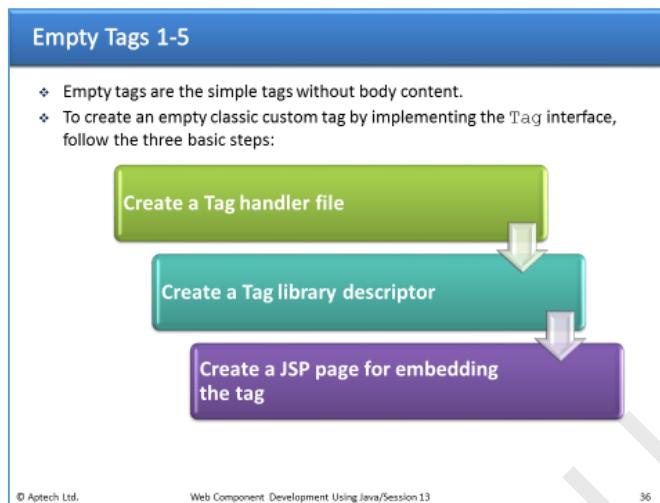
Tell the students `doStartTag()` is defined by a `Tag` interface that is called by the JSP page's servlet when a start tag is evaluated. This function returns static final integer constant value that is defined in the interface.

Using slides 33 and 34, explain the methods of `Tag` interface.

Explain the code snippet as shown on slide 35 to demonstrate the methods of `Tag` interface.

Slides 36 to 40

Let us understand empty tags.



Empty Tags 4-5

- ❖ **Create a Tag library descriptor**
 - We will create entry in tag library descriptor file for the empty.
 - The code snippet shows TLD file that will provide a detailed description of the Tag library and the custom tag.

```
<tag>
<name>Implement</name>
<tag-class>mytag.ImplementTag</tag-class>
<body-content>empty</body-content>
</tag>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 39

Empty Tags 5-5

- ❖ **Create a JSP page for embedding the tag**
 - In the final step, the code snippet shows that tag needs to be embedded in a JSP file.

```
<%--The tag is imported by the following directive.--%
<%@taglib uri="/WEB-INF/tlds/TagInterface.tld"
prefix="taginterface"%><HTML>
<HEAD></HEAD>
<BODY>
<%--The custom tag is called here.--%>
<taginterface:ImplementTag></taginterface:Implement>
</BODY>
</HTML>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 40

Use slides 36 to 40 to explain Empty tags.

Tell them using Tag interface, you can create an empty tag with or without attributes.

Tell the students that a Custom tag is called an empty tag when it doesn't contain any attribute or body.

To create an empty tag using tag interface, the steps to be performed are as follows:

1. Create a tag handler class
2. Create a tag library descriptor defining the tag handler class
3. Create a JSP page to embed the custom tag in it

Using slides 37 to 40, explain each of the steps to create a tag handler class named `ImplementTag` and accessing it on JSP page.

Slides 41 to 44

Let us understand how to create empty tag to accept attribute.

Empty Tag to Accept Attribute 1-4

- ❖ The steps to create a classic custom tag to accept attribute are as follows:
 - Create a tag handler file
 - The tag handler class for the classic custom tag that accepts attribute need to implement the Tag interface.
 - Attributes are passed from the tag handler class to the tag inside the JSP page.
 - Only the start and end tags of the custom tag needs to be processed.
 - The code snippet demonstrates how to define attributes for the empty tag.

```
//The Tag interface is implemented to the TagAttribute class.
public class TagAttribute implements Tag {
    private String name;
    //The attribute .name is set by the setter method, setName()
    public void setName(String name) {
        this.name = name;
    }
    // The pageContext instance is set by the setPageContext() method
    public void setPageContext(PageContext pc) {
        this.pageContext = pc;
    }
    //The instance of the parent tag is retrieved here.
    public Tag getParent() {
        return parent;
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 13 41

Empty Tag to Accept Attribute 2-4

```
//The evaluation of start tag starts
public int doStartTag() throws JspException {
    try {
        pageContext.getOut().print(
            "This is my first tag with attribute!" + name);
    } catch (IOException ioe) {
        throw new JspException("Error: "
            + ioe.getMessage());
    }
    return SKIP_BODY;
}
//The evaluation of end tag starts
public int doEndTag() throws JspException {
    return SKIP_PAGE;
}
//All resources held by the tag handler is released
public void release() {
}
```

© Aptech Ltd. Web Component Development Using Java/Session 13 42

Empty Tag to Accept Attribute 3-4

- Create a Tag library descriptor
 - We will create entry in tag library descriptor file for the empty.
 - The TLD file will provide a detailed description of the Tag library and the custom tag along with the attributes passed to it within the xml elements.
 - The code snippet shows the tag library descriptor.

```
<tag>
<name>welcomparam</name>
<tagclass>tags.TagAttribute</tagclass>
<bodycontent>empty</bodycontent>
<attribute>
<name>name</name>
<required>false</required>
<rteprvalue>false</rteprvalue>
</attribute>
</tag>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 43

Empty Tag to Accept Attribute 4-4

□ Create a JSP page for embedding the tag

- In the final step, the code snippet tag needs to be embedded in a JSP file.

```
<%--The tag is imported by the following  
directive.--%>  
<%@ taglib uri="/WEB-INF/jsp/TagAttribute.tld"  
prefix="first" %>  
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
<%-- The tag is declared --%>  
<first:welcomparam name="APTECH"/>  
</BODY>  
</HTML>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 44

Use slides 41 to 44 to explain how to create empty tag to accept attribute.

Tell them that in the last few slides, you have seen how to create an empty tag handler class. You can also create an empty handler with attributes. Tags that have attributes are also called parameterized tags.

Then, tell them that a classic custom tag handler can be created in the following steps:

- Create a tag handler file
- Create a tag library descriptor
- Create a JSP page for embedding the tag

Using slides 41 and 42, explain the code for defining the attributes for the empty tag. Then, using slide 43, explain the tag library descriptor for defining the tag. Finally, using slide 44, explain how to embed the tag in the JSP page.

Slides 45 to 48

Let us understand Non-Empty tags.

Non-empty Tags 1-4

- ❖ Tags that include body inside them are non-empty tags.
- ❖ We need to follow the three basic steps of creating a custom tag that has a body:
 - **Create a tag handler file:**
 - The tag handler class for the classic custom tag that has body content implements the Tag interface.
 - The doStartTag() method returns EVAL_BODY_INCLUDE.
 - The body is evaluated by the JSP container as any other java codes in the JSP page.

© Aptech Ltd. Web Component Development Using Java/Session 13 45

Non-empty Tags 2-4

- ❖ The code snippet demonstrates how to create the handler for the non-empty tag.

```
public class MyBodyTag implements Tag {
    private String name=null;
    private PageContext pc;
    private Tag pt;
    public void setName(String value) {
        name = value;
    }
    public String getName(){
        return(name);
    }
    public void setPageContext(PageContext pageContext) {
        this.pc=pageContext;
    }
    public void setParent(Tag tag) {
        this.pt=tag;
    }
    public Tag getParent() {
        return pt;
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 13 46

Non-empty Tags 3-4

- **Create a Tag library descriptor**
 - The TLD file in this case will have the <body-content> as JSP, as shown in the code snippet.

```
<tag>
<name> MyBodyTag </name>
<tag-class>mytag.MyBodyTag </tag-class>
<body-content>JSP</body-content>
</tag>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 47

Non-empty Tags 4-4

□ Create a JSP page for embedding the tag

- In the final step, the tag needs to be embedded in a JSP file, as shown in the code snippet.

```
<%--The tag is imported by the following directive.--%>
<%@taglib uri="/WEB-INF/tlds/UsingTag.tld"
prefix="usingtag"%>
<HTML>
<HEAD>
<TITLE>Tag with Body</TITLE>
</HEAD>
<BODY>
    <%-- The tag is declared --%>
    <usingtag:Interface>
        <br>Current time: <%= new java.util.Date() %>
    </usingtag:MyBodyTag>
</BODY>
</HTML>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 48

Using slides 45 to 48, explain how to create non-empty tags.

The steps to be followed for creating non-empty tags are:

- Create a tag handler class
- Create a Tag library descriptor
- Create a JSP page for embedding the tag

Using slide 46, explain how to create the handler class for the non-empty tag, MyBodyTag. Then, using slides 47 and 48, explain the tag descriptor and JSP page to access the tag.

Slides 49 and 50

Let us understand the methods of `IterationTag` Interface.

Methods of IterationTag Interface

- ❖ This interface has the method `doAfterBody()`.
- ❖ `doAfterBody()`:
 - ❑ Allows the user to conditionally re-evaluate the body of the tag.
 - ❑ Returns the constant `SKIP_BODY` or `EVAL_BODY_AGAIN`, if the `doStartTag()` method returns `EVAL_BODY_INCLUDE`, then this method returns `EVAL_BODY_AGAIN` and the `doStartTag()` is evaluated once again.
 - ❑ When the `doAfterBody()` returns `SKIP_PAGE`, the `doEndTag()` method is invoked.
 - ❑ **Syntax:**

```
public int doAfterBody() throws JspException
```

© Aptech Ltd.

Web Component Development Using Java/Session 13

49

Iterative Tag Sample

- ❖ The code snippet shows the `IterationTag` in a JSP page.

```
<%@ taglib prefix="iteration" uri="/WEB  
INF/sampleIterationTagLib.tld" %>  
<html><body>  
  <iteration:loop count="5" >  
    Hello IterationTag!<br>  
  </iteration:loop>  
</body></html>
```

© Aptech Ltd.

Web Component Development Using Java/Session 13

50

Use slides 49 and 50 to explain the methods of `IterationTag` interface. Tell the students that using `doAfterBody()` method, you can iterate the body content of any tag.

The `doAfterBody()` method allows the user to conditionally re-evaluate the body of the tag. If `doAfterBody()` returns `IterationTag.EVAL_BODY_AGAIN`, then the body will be reevaluated. If it returns `Tag.SKIP_BODY`, then the body will be skipped and `doEndTag()` will be evaluated.

Use slide 50 to explain the code snippet that shows an iterative tag sample for accessing the iterator tag on JSP page.

Slides 51 and 52

Let us understand the methods of BodyTag interface.

Methods of BodyTag Interface

- ❖ The methods inside the BodyTag interface are as follows:
 - `setBodyContent ()`
 - Sets the `bodyContent` object for the tag handler.
 - Encapsulates the body content of the custom tag for processing.
 - Is automatically invoked by the JSP page before the `doInitBody ()` method.
 - **Syntax:**

```
public void setBodyContent(BodyContent b){ }
```
 - `doInitBody ()`
 - Is invoked immediately after the `setBodyContent ()` method returns the `bodyContent` object and before the first body evaluation.
 - **Syntax:**

```
public int doAfterBody() throws JspException
```

© Aptech Ltd. Web Component Development Using Java/Session 13 51

Sample BodyTag

- ❖ The code snippet shows the BodyTag in a JSP page.


```
<<%@ taglib prefix="bodyTagPrefix" uri="/WEB-INF/sampleBodyTagLib.tld" %>
<html>
<body>
<bodyTagPrefix:convertTempconvert="C" >
Temperature is:<%=temp%></br>
</ bodyTagPrefix:loop>
</body>
</html>
```

© Aptech Ltd. Web Component Development Using Java/Session 13 52

Use slides 51 and 52 to explain the methods of BodyTag Interface.

Tell the students the BodyTag interface extends IterationTag by defining additional methods.

Use slide 52 to explain a sample BodyTag code snippet.

Slide 53

Let us understand extending TagSupport and BodyTagSupport.

Extending TagSupport and BodyTagSupport

- ❖ The tag handler class needs to implement all the method declarations of the Tag or IterationTag interface when implementing these interfaces.
- ❖ In some case, we do not required to implement all these methods.
- ❖ The TagSupport class, which implements the Tag or IterationTag interface and provides a default implementation of all the methods.
- ❖ The programmer can extend TagSupport class and override the required method.

© Aptech Ltd. Web Component Development Using Java/Session 13 53

Use slide 53 to explain extending TagSupport and BodyTagSupport tag.

The tag handler class needs to implement all the method declarations of the Tag or IterationTag interface when implementing these interfaces. In some case, we do not required to implement all these methods. The TagSupport class, which implements the Tag or IterationTag interface and provide a default implementation of all the methods. The programmer can extend TagSupport class and override the required method.

Slides 54 to 57

Let us understand TagSupport class.

TagSupport Class 1-2

- ❖ The TagSupport class implements the IterationTag interface.
- ❖ It provides default implementations for each of the methods of the Tag and IterationTag interfaces.
- ❖ Some methods of TagSupport class are as follows:
 - doStartTag() : This method is inherited from Tag interface and by default, return value is SKIP_BODY.
 - doEndTag() : This method is inherited from IterationTag interface and by default, return value is SKIP_BODY.

© Aptech Ltd. Web Component Development Using Java/Session 13 54

TagSupport Class 2-2

- ❖ Figure depicts the hierarchy of TagSupport class.

```

graph TD
    TagInterface[Tag Interface] --> IterationTagInterface[IterationTag Interface]
    IterationTagInterface --> TagSupportClass([TagSupport Class])
    TagSupportClass --> doStartTag[doStartTag()]
    TagSupportClass --> doEndTag[doEndTag()]
  
```

© Aptech Ltd. Web Component Development Using Java/Session 13 55

BodyTagSupport Class 1-2

- ❖ This class by default implements the BodyTag interfaces and also extends to the TagSupport class.
- ❖ The handler class only needs to override the methods of the BodyTagSupport class.
- ❖ All the methods of the TagSupport class and BodyTag interface are implemented by default in the handler class.
- ❖ Figure depicts the hierarchy of BodyTagSupport class.

```

graph TD
    BodyTagInterface[BodyTag Interface] --> BodyTagSupportClass([BodyTagSupport class])
    BodyTagSupportClass --> BodyTagSupportClass
  
```

© Aptech Ltd. Web Component Development Using Java/Session 13 56

BodyTagSupport Class 2-2

- ❖ The BodyTagSupport class contains the following overridden methods:

- **getBodyContent()** :

- Retrieves the bodyContent object of the class BodyContent.
 - Contains the body of the tag which is accessed by the container for manipulation.

- **Syntax:**

```
public BodyContent getBodyContent()
```

- **setBodyContent()** :

- Is set by the `setBodyContent()` method.

- **Syntax:**

```
public void setBodyContent(BodyContent b)
```

Use slides 54 to 57 to explain TagSupport class and BodyTagSupport class.

Tell the students that the TagSupport class is used as the base class for new tag handlers. The TagSupport class implements the Tag and IterationTag interfaces and adds additional convenience methods including getter methods for the properties in tag.

Use slide 55 to explain the hierarchy of TagSupport class.

Then, using slides 56 and 57, explain BodyTagSupport class. Tell the students that BodyTagSupport is a class of package `javax.servlet.jsp.tagext` that extends the class TagSupport and implements the BodyTag interface. This class is used as a base class to pass attributes.

The BodyTagSupport class implements the BodyTag interface and adds additional convenience methods including getter methods for the `bodyContent` property and methods to get at the previous out `JspWriter`. Explain the methods of BodyTagSupport class as mentioned on slide 57.

Slides 58 to 60

Let us understand SimpleTag.

SimpleTag 1-3

- Figure depicts the relationship between SimpleTag and other JSP tag interfaces.

```

graph TD
    JSPTag[JSP Tag] --> Tag[Tag]
    JSPTag --> SimpleTag[Simple Tag]
    Tag --> BodyTag[Body Tag]
    Tag --> IterationTag[Iteration Tag]
  
```

© Aptech Ltd. Web Component Development Using Java/Session13 58

SimpleTag 2-3

- Following methods are present in SimpleTag interface:

- doTag()**
 - Is called by the container to begin SimpleTag operation and used for handling tag processing.
 - Retains body iteration after being processed.
 - Syntax:
`public void doTag() throws JspException, IOException`
- setParent()**
 - Sets the parent of a specified tag.
 - Syntax:
`public void setParent(JspTag parent)`
- getParent()**
 - Returns the parent of the specified tag.
 - Syntax:
`public JspTag getParent()`

© Aptech Ltd. Web Component Development Using Java/Session13 59

SimpleTag 3-3

- setJspContext()**
 - Sets the context to the tag handler for invocation by the container.
 - Syntax:
`public void setJspContext(JspContext pc)`
- setJspBody()**
 - Is provided by the body of the specified tag, makes the body content available for tag processing.
 - Syntax:
`public void setJspBody(JspFragment jspBody)`

© Aptech Ltd. Web Component Development Using Java/Session13 60

Use slides 58 to 60 to explain SimpleTag.

Tell them that you can create the tag handler class of type SimpleTag.

Tell them implement the `javax.servlet.jsp.tagext.SimpleTag` interface, if they want to have a simpler invocation protocol. The `SimpleTag` interface does not extend the `javax.servlet.jsp.tagext.Tag` interface as does the `BodyTag` interface. Therefore, instead of supporting the `doStartTag()` and `doEndTag()` methods, the `SimpleTag` interface provides a simple `doTag()` method, which is called once for each tag invocation.

Tell the students that simple tag handlers are never cached and reused by the JSP container.

Some of the other methods of `SimpleTag` interface are as explained:

The `setJspContext()` and `setParent()` methods are called by the container. The `setParent()` method is only called if the element is nested within another tag invocation. The setters for each attribute defined for this tag are called by the container.

If a body exists, the `setJspBody()` method is called by the container to set the body of this tag, as a `JspFragment`. If the action element is empty in the page, this method is not called at all. The `doTag()` method is called by the container. All tag logic, iteration, and body evaluations, occur in this method. The `doTag()` method returns and all variables are synchronized.

Slides 61 to 64

Let us understand the implementation of SimpleTagSupport class.

SimpleTagSupport

- ❖ The SimpleTagSupport class acts as a base class for simple tag handlers.
- ❖ The class implements the SimpleTag interface.
- ❖ Some of the useful methods are as follows:
 - **getJspContext()**
 - Returns the context passed into the container by setJspContext() method.
 - **Syntax:**

```
protected JspContext getJspContext()
```
- **getJspBody()**
 - Returns the JspFragment object for body processing in the tag.
 - **Syntax:**

```
protected JspFragment getJspBody()
```

© Aptech Ltd.

Web Component Development Using Java/Session 13

61

Implementation of SimpleTag Interface 1-3

- ❖ The code snippet shows the interface that is implemented by extending the SimpleTagSupport class and overriding the doTag() method.

```
import javax.servlet.jsp.tagext.*;
import javax.servlet.jsp.*;
public class Greeter extends SimpleTagSupport {
    public void doTag() throws JspException {
        PageContext pageContext = (PageContext)
        getJspContext();
        JspWriter out = pageContext.getOut();
    }
}
```

© Aptech Ltd.

Web Component Development Using Java/Session 13

62

Implementation of SimpleTag Interface 2-3

- ❖ The code snippet shows the corresponding tag in TLD file.

```
<tag>
<name>greeter</name>
<tag-class>Greeter</tag-class>
<body-content>empty</body-content>
<description>
    Print this on the browser.
</description>
</tag>
```

© Aptech Ltd.

Web Component Development Using Java/Session 13

63

Implementation of SimpleTag Interface 3-3

- ❖ The code snippet shows the corresponding JSP file.

```
<%@ taglib prefix="ui" uri="/WEB-INF/sampleTag.tld" %>
<html><body>
<ui: greeter />
</body></html>
```

Use slides 61 to 64 to explain SimpleTagSupport class.

Tell the students the SimpleTagSupport class is used as the base class for new simple tag handlers. The SimpleTagSupport class implements the SimpleTag interface and adds additional convenience methods including getter methods for the properties in SimpleTag.

Use slides 62 to 64 to explain the implementation of SimpleTag interface.

Slide 65

Let us summarize the session.

Summary

- ❖ Custom tags are action elements on JSP pages that are mapped to tag handler classes in a tag library.
- ❖ Tag libraries allow us to use independent Java classes to manage the presentation logic of the JSP pages, thereby reducing the use of scriptlets and leveraging existing code to accelerate development time.
- ❖ The tag without any body is called an empty tag and the tag accepting attributes is called a tag with attributes or a parameterized tag.
- ❖ A tag with JSP code or tag dependent strings is called a tag with body. If a tag contains several other tags inside it then it is called a nested tag.
- ❖ The Tag Library Descriptor (TLD) file contains the information that the JSP engine needs to know about the tag library in order to successfully interpret the custom tags on JSP pages.
- ❖ The SimpleTag interface provides the doTag() method, which is supported by the Classic Tag Handlers.

© Aptech Ltd. Web Component Development Using Java/Session 13 65

In slide 65, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

13.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should explore the concept of internationalization and localization for displaying the Web applications based on geographical locations.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 14 – Internationalization

14.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

14.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the concept of internationalization
- Explain the concept of localization
- Explain the role of Unicode character set in internationalization
- Explain the resource bundling mechanism and resource bundle for various locales
- Explain how to format dates in servlets for internationalization
- Explain how to format currency in servlets for internationalization
- Explain how to format numbers in servlets for internationalization
- Explain how to format percentages in servlets for internationalization
- Explain how to format messages in servlets for internationalization
- Explain various tags available in the JSTL internationalization tag library
- Explain how to format dates and currencies using JSTL I18N tags
- Explain how to format percentages and messages using JSTL I18N tags

14.1.2 Teaching Skills

To teach this session successfully, you should know about the concepts of internationalization and localization. Familiarize yourself with Unicode character set and resource bundling mechanism. Aware yourself with internationalizing servlets by formatting dates, currency, numbers, percentages, and messages based on geographical locations.

You should be aware of various tags available in JSTL internationalization tag library. Finally, you should also know how to format dates currencies, percentages, and messages using JSTL I18N tags.

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❖ Explain the concept of internationalization
- ❖ Explain the concept of localization
- ❖ Explain the role of Unicode character set in internationalization
- ❖ Explain the resource bundling mechanism and resource bundle for various locales
- ❖ Explain how to format dates in servlets for internationalization
- ❖ Explain how to format currency in servlets for internationalization
- ❖ Explain how to format numbers in servlets for internationalization
- ❖ Explain how to format percentages in servlets for internationalization
- ❖ Explain how to format messages in servlets for internationalization
- ❖ Explain various tags available in the JSTL internationalization tag library
- ❖ Explain how to format dates and currencies using JSTL I18N tags
- ❖ Explain how to format percentages and messages using JSTL I18N tags

© Aptech Ltd. Web Component Development Using Java/Session 14 2

Tell the students that they will be introduced to the concepts of internationalization and localization. Explain Unicode character set and resource bundling mechanism. In this session, they will learn how to display servlets based on their geographical locations. This is termed as internationalization servlets.

Explain various tags available in JSTL internationalization tag library. Finally, discuss how to format dates currencies, percentages, and messages using JSTL I18N tags.

14.2 In-Class Explanations

Slide 3

Let us understand internationalization.



Use slide 3 to explain internationalization.

An application is accessible to the international market when the input and output operations are specific to different locations and user preferences. The process of designing such an application is called internationalization.

Internationalization is the process of developing an application that can be adapted to various languages and regions without engineering changes. In fact, Internationalization covers more than just language. It also covers formatting of numbers, date and time, and so on.

Internationalization is commonly referred to as i18n. 18 in i18n refer to the 18 characters between the first letter i and the last letter n. Java includes a built-in support to internationalize applications.

Tips:

An internationalized program has the following characteristics:

With the addition of localized data, the same executable can run worldwide.

- Textual elements, such as status messages and the GUI component labels, are not hardcoded in the program. Instead, they are stored outside the source code and retrieved dynamically.
- Support for new languages which does not require recompilation.
- Culturally-dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.
- It can be localized quickly.

In-Class Question:

After you finish explaining Internationalization, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which mechanism can be used for designing an application that can be adapted to a region or a language without much change in the technology?

Answer:

Internationalization

Slide 4

Let us understand localization.

Localization

- ❖ Localization is a process of making a product or service language, culture, and local 'look-and-feel' specific.
- ❖ Localizing a product does not only mean a language translation, but also formatting of time, currencies, messages, and dates.
- ❖ In Java:
 - ❑ A `Locale` is a simple object, which identifies a specific language and a geographic region.
 - ❑ To create international Java applications, the use of `java.util.Locale` class is a must.
 - ❑ Locales are used in entire Java class libraries data for formatting and customization.



© Aptech Ltd. Web Component Development Using Java/Session 14 4

Use slide 4 to explain localization.

Localization deals with a specific region or language. In localization, an application is adapted to a specific region or language. Locale-specific components are added and text is translated in the localization process.

Localization is what the Java application does when it adapts itself to a user with a specific language, number format, date and time, and so on.

Localization is commonly referred as l10n. 10 in l10n refers to the 10 letters between the first letter l and the last letter n. Primarily, in localization, the user interface elements and documentation are translated.

In-Class Question:

After you finish explaining localization, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which class is a simple object that identifies a specific language and a geographic region?

Answer:

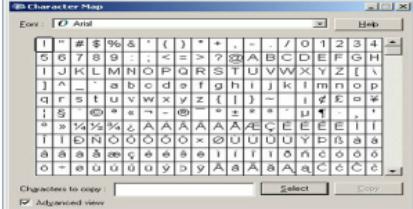
Locale

Slide 5

Let us understand Unicode.

Unicode

- ❖ Is a coding system that has codes for all the major language of the world.
- ❖ Is a 16-bit character encoding.
- ❖ Uses UCS-2: a fixed-width two byte encoding that simply encodes each code point from U+0000 to U+FFFF as itself.
- ❖ Allows Java to handle international characters for most of the languages of the world.
- ❖ Figure depicts the symbol table supported in Unicode.



© Aptech Ltd. Web Component Development Using Java/Session 14 5

Use slide 5 to explain Unicode.

Unicode is a computing industry standard for the consistent encoding, representation, and handling a text expressed in most of the world writing system. Unicode success at unifying character sets has led to its widespread and predominant use in the internationalization and localization of computer software. The standard has been implemented in many recent technologies, including XML, Java programming language, and modern operating system.

Unicode provides a unique number for every character irrespective of platform, program, or language. The Unicode standard was first designed using 16 bits to encode characters. 16-bit encoding supports 2¹⁶ (65,536) characters where in the hexadecimal, they ranged from 0x0000 to 0xFFFF.

Tell the students that Unicode standard uses hexadecimal to express a character. For example, the value 0x0032 represents the number '2'. Unicode standard supports over one million characters.

Slide 6

Let us understand Locale.

The slide has a title 'Locale' at the top. Below it is a bulleted list of four points:

- Locale represent specific geographical, political, or cultural region.
- Locale is denoted by the standard format `xx_YY`, where, `xx` stands for two-letter language code in lower case, and `YY` stands for two-letter country code in upper case.
- Some of the examples of locales are `zh_CN` for Shanghai, China; `ko_KR` for Seoul, Korea; and `en_US` for English, US.
- Figure depicts the use of various locales.

Below the list are three screenshots of a Java application window titled 'SimpleMenu: English (United States)'. The window contains three tabs: 'Colors', 'Colours', and 'Couleurs'. Each tab lists three colors: Red, Green, and Blue, with their corresponding names in English, French, and German respectively. The tabs are labeled 'Colors', 'Colours', and 'Couleurs' respectively, indicating the locale settings for each tab.

At the bottom left is the copyright notice '© Aptech Ltd.', and at the bottom center is the page number 'Web Component Development Using Java/Session 14'.

Use slide 6 to explain Locale.

Tell the students a locale represents a specific geographical, political, or cultural region. An operation that needs a locale to carry out its task is called locale-sensitive.

A `Locale` object consists of fields namely, language, script, variant, and extensions. A `Locale` object can be created using Builder, Constructors, and Constants.

➤ Builder

Using `Locale.Builder`, user can create a `Locale` object that conforms to BCP 47 syntax.

➤ Constructors

The three constructors for creating a `Locale` object are:

- `Locale(String language)`
- `Locale(String language, String country)`
- `Locale(String language, String country, String variant)`

These constructors helps to create a `Locale` object with language, country, and variant.

➤ Locale Constants

The `Locale` class has a number of constants that can be used to create `Locale` objects for commonly used locales. For example, the following creates a `Locale` object for the South Africa: `Locale.RSA`

Tips:

The following servlet code displays Hello World in Japanese language using Locale object:

```
public class JapneseHelloWorldServlet extends HttpServlet {  
  
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
  
    response.setContentType("text/html; charset=UTF-8");  
  
    PrintWriter out = response.getWriter();  
  
    response.setHeader("Content-Language", "ja");  
  
    // Locale object with language set as ja for japanese  
  
    Locale locale = new Locale("ja", "");  
  
    out.println("In Japanese:");  
  
    // Unicode characters for Japanese Hello World  
  
    out.println("\u4eca\u65e5\u306f\u4e16\u754c");  
  
}  
}
```

Slides 7 and 8

Let us understand how to create a ResourceBundle.

Creating a ResourceBundle 1-2

How to create a internalization of Web application in Java?



ANSWERS

Create text files with key/value pairs representing text from different locales and store them under WEB-INF folder.

- ❖ For example, the key “Welcome” is associated with a value or message in different ResourceBundle property files.
 - Key – Represents the word.
 - Value – Represents the translation in the specific locale.

© Aptech Ltd. Web Component Development Using Java/Session 14 7

Creating a ResourceBundle 2-2

- ❖ The code snippet shows the ResourceBundle property file named DemoResources_es_ES.properties containing Spanish equivalent for the key “Welcome”.


```
#Spanish language resources
Welcome = Hola y recepción
```
- ❖ The code snippet shows the ResourceBundle property file named DemoResources_en_US.properties containing United States English equivalent for the key “Welcome”.


```
#English language resources
Welcome = Hello and welcome
```

© Aptech Ltd. Web Component Development Using Java/Session 14 8

Use slides 7 and 8 to explain how to create a ResourceBundle.

The resource bundling helps to build server-side code that gives output based on the location and language of the user. It makes the job easier by avoiding the writing of multiple version of a class for different locales.

A **ResourceBundle** is a property file containing a list of keys and values. It can be created using a text editor. The keys represents the words that you want to be translated, and the values are the translations. These files are the resources that the Web application uses to dynamically translate text into the appropriate language.

Then, explain them the code snippet given on slide 8 for creating the resource bundles for the visitors from the locale en_US for United State English and also from the locale es_ES for Spanish.

Slides 9 to 14

Let us understand internationalizing servlets.

Internationalizing Servlets 1-6

- ❖ Internationalization of an application can be achieved with the help of resource bundles.
- ❖ **Resource Bundle:**
 - ❑ Is a set of related classes that inherit from the ResourceBundle.
 - ❑ The subclass of ResourceBundle has the same base name with an additional component that identifies locales.
 - ❑ For example, if resource bundle is named DemoResources and along with it, locale-specific classes can be related as in DemoResources_en_US.
 - ❑ Helps to build server-side code that gives output based on the location and language of the user.
 - ❑ Makes the job easier by avoiding the writing of multiple version of a class for different locales.

© Aptech Ltd. Web Component Development Using Java/Session 14 9

Internationalizing Servlets 2-6

- ❖ There are several methods in ResourceBundle class.

```
public static final ResourceBundle getBundle(String
baseName)
```

- ❑ In order to get resource bundle for getting the locale-specific data, the getBundle() method is used.
- ❑ This method gets a resource bundle using the specified base name, the default locale, and the class loader of the caller.

- ❖ The code snippet demonstrates the method to get a resource bundle using the specified base name and locale.

```
/* This snippet creates ResourceBundle by invoking
getBundle() method, specifying the base name */
ResourceBundle labels =
ResourceBundle.getBundle("DemoLabelsBundle");
```

© Aptech Ltd. Web Component Development Using Java/Session 14 10

Internationalizing Servlets 3-6

```
public abstract Enumeration getKeys()
```

- ❑ Returns an enumeration of the keys present in the property file.

- ❖ The code snippet shows the use of the getKeys() method.

```
// this snippet displays the value of the keys
static void iterateKeys(Locale currentLocale) {
    ResourceBundle labels =
    ResourceBundle.getBundle("LabelsBundle",currentLocale);
    Enumeration bundleKeys = labels.getKeys();
    while (bundleKeys.hasMoreElements()) {
        String key = (String)bundleKeys.nextElement();
        String value = labels.getString(key);
        System.out.println("key = " + key + ", " +
                           "value = " + value);
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 14 11

Internationalizing Servlets 4-6

public Locale getLocale()

- ❑ Returns the locale of this resource bundle.
- ❑ Can be used after the calling of `getBundle()` to check whether the returned resource bundle really corresponds to the requested locale or not.
- ❑ Returns the locale for the current resource bundle.

❖ The code snippet shows the use of the `getLocale()` method.

```
// This snippet gets the user's Locale
Locale locale = request.getLocale();
ResourceBundle bundle =
ResourceBundle.getBundle("i18n.WelcomeDemoBundle",
locale);
String welcome = bundle.getString("DemoWelcome");
```

© Aptech Ltd. Web Component Development Using Java/Session 14 12

Internationalizing Servlets 5-6

public final Object getObject(String key)

- Gets an object for the given key from the resource bundle.
- Gets the object from the resource bundle and if it fails, the parent resource bundle is called using parent's `getObject()` method.
- Throws a `MissingResourceException` if it fails.
- For example, `int[] myDemoIntegers = (int[])
myDemoResources.getObject ("intList");`

© Aptech Ltd. Web Component Development Using Java/Session 14 13

Internationalizing Servlets 6-6

public final String getString(String key)

- ❑ Returns a string for the given key from the resource bundle or one of its parents.

❖ The code snippet shows the use of `getString()` method.

```
/** Retrieves the translated value from the
ResourceBundle by invoking the getString method as
follows */
String value = labels.getString(key);
```

© Aptech Ltd. Web Component Development Using Java/Session 14 14

Use slides 9 to 14 to explain how to internationalize Servlets. Tell the students that the `ResourceBundle` class is used to internationalize the messages in the Web application.

Some of the most commonly used methods of ResourceBundle class are as follows:

- `static ResourceBundle getBundle(String basename)` - returns the instance of the ResourceBundle class for the default locale.
- `static ResourceBundle getBundle(String basename, Locale locale)` - returns the instance of the ResourceBundle class for the specified locale.
- `String getString(String key)` - returns the value for the corresponding key from the resource bundle.
- `static void clearCache()` - removes all resource bundles from the cache that have been loaded using the caller's class loader.
- `static ResourceBundle getBundle(String baseName, ResourceBundle.Control control)` - returns a resource bundle using the specified base name, the default locale and the specified control.
- `boolean containsKey(String key)` - determines whether the given key is contained in the ResourceBundle or its parent bundles.
- `Object getObject(String key)` - gets an object for the given key from the resource bundle or one of its parents.
- `protected abstract Object handleGetObject(String key)` - gets an object for the given key from the resource bundle.
- `Set<String> keySet()` - returns a set of all keys contained in the ResourceBundle and its parent bundles.
- `String[] getStringArray(String key)` - gets a string array for the given key from the resource bundle or one of its parents.

Tips:

To develop a complete servlet using ResourceBundle class, you can refer this link:

<http://www.devmanuals.com/tutorials/java/servlet/servlet-internationalization.html>

In-Class Question:

After you finish explaining internationalize Servlets, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



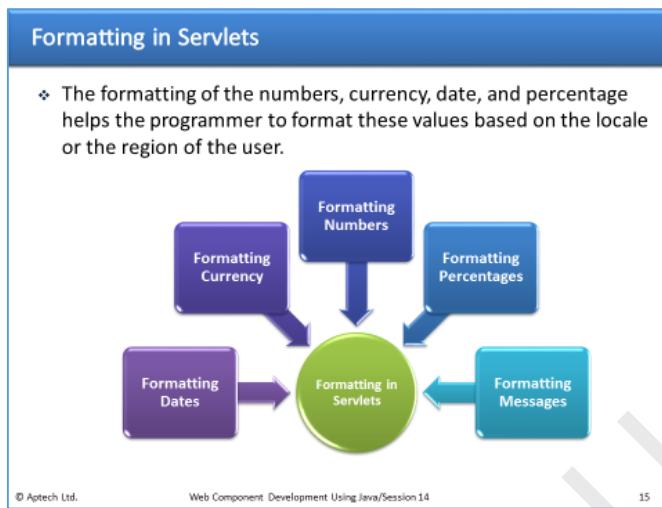
Which method determines whether the given key is contained in the ResourceBundle or its parent bundles?

Answer:

`boolean containsKey(String key)`

Slide 15

Let us understand formatting in Servlets.



Use slide 15 to explain formatting in Servlets.

Tell them formatting of the numbers, currency, date, and percentage helps the programmer to format these values based on the locale or the region of the user.

Slides 16 to 19

Let us understand formatting dates.

Formatting Dates 1-4

- The formats in which dates can be displayed include **Predefined Formats** and **Customizing Formats**.

Predefined Formats

- The **DateFormat** style is predefined and locale-specific and is easy to use. The styles are as follows:
 - SHORT** - It is completely numeric, such as 11.14.50 or 3:30pm.
 - MEDIUM** - it is longer, such as Jan 10, 1954.
 - LONG** - it is longer, such as January 10, 1954 or 3:50:32pm.
 - FULL** - it is completely specified, such as Tuesday, April 14, 1954 AD or 3:50:42pm PST.
- The two steps in formatting of date using the **DateFormat** class are as follows:

- Creating a formatter with the `getDateInstance()` method
- Invoking the `format()` method, which returns formatted date in the form of string

© Aptech Ltd. Web Component Development Using Java/Session 14 16

Formatting Dates 2-4

Customizing Formats

- Most of the time, the predefined formats are enough, but sometimes customized format is required.
- To achieve custom formatting, the **SimpleDateFormat** class is used.
- SimpleDateFormat class:**
 - Formats and parses dates in a locale-sensitive manner.
 - The code snippet demonstrates the use of the **SimpleDateFormat** class.

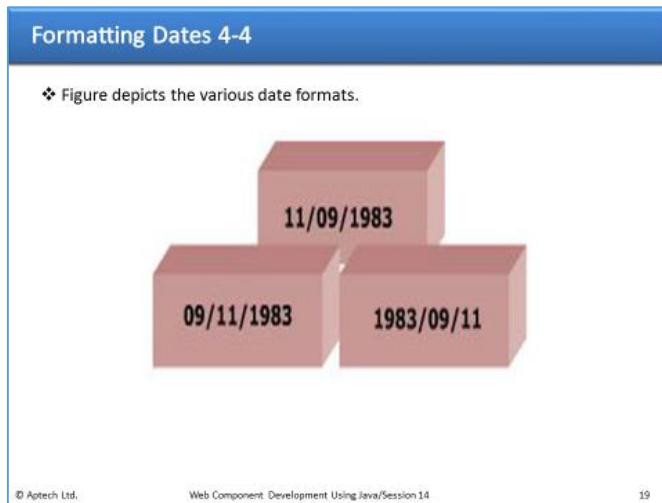
```
/* this snippet formats the date as per the
specified style */
SimpleDateFormat fmt = new SimpleDateFormat("yyyy-
MM-dd", Locale.US);
```

© Aptech Ltd. Web Component Development Using Java/Session 14 17

Formatting Dates 3-4

- The several classes for formatting dates are as follows:
 - DateFormat** - Formats and parses the dates and time in a language independent manner.
 - DateFormat.Field** - Is a nested class that is used as attribute keys in the **AttributedCharacterIterator**.
 - DateFormatSymbol** - Encapsulates localizable date-time formatting data, such as names of months, days of week, and the time zone data.
 - DateFormatter** - Formats the date as per the format of the current locale.

© Aptech Ltd. Web Component Development Using Java/Session 14 18



Use slides 16 to 19 to explain formatting dates.

The `java.text.DateFormat` class is used to format dates as strings according to a specific locale. A date format can be created using the `getDateInstance()` and `getTimeInstance()` method of the `DateFormat` class.

The date format constants in the `DateFormat` class are as follows:

- `DateFormat.DEFAULT`
- `DateFormat.SHORT`
- `DateFormat.MEDIUM`
- `DateFormat.LONG`
- `DateFormat.FULL`

The following additional code snippet shows the use of `getDateInstance()` method to display date of different countries:

```
public class DateServlet extends HttpServlet {
{
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

Date date = new Date();

DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT,
Locale.KOREA);

out.println("KOREA: " + df.format(date));
}
}
```

```
df = DateFormat.getDateInstance(DateFormat.MEDIUM, Locale.JAPAN);
System.out.println("JAPAN: " + df.format(date));

df = DateFormat.getDateInstance(DateFormat.LONG, Locale.UK);
System.out.println("United Kingdom: " + df.format(date));

df = DateFormat.getDateInstance(DateFormat.FULL, Locale.US);
System.out.println("United States: " + df.format(date));

}
```

Then, tell them that most of the time, the predefined formats are enough but sometime customized format is required. To do that `SimpleDateFormat` class is used. A `SimpleDateFormat` object is created with specified pattern strings.

The following additional code snippet shows how to work with `SimpleDateFormat` class.

```
...
Date date = new Date();

SimpleDateFormat format = new SimpleDateFormat("yyyy/MM/dd");

String DateToStr = format.format(date);
out.println(DateToStr);

format = new SimpleDateFormat("dd-M-yyyy hh:mm:ss");
DateToStr = format.format(date);
System.out.println(DateToStr);

...
```

Slides 20 to 23

Let us understand formatting currency.

Formatting Currency 1-4

- ❖ The formatting of the currency is necessary, so that the users can be benefited by the availability of the locale specific formatting.
- ❖ Conversion of one currency to another requires some conversion in value.
- ❖ The formatting of currency value is much more involved compared to formatting date and time.
- ❖ Some of the classes used for formatting currency are as follows:

Currency	• This class represents currency. The currencies are identified by ISO 4217 currency codes.
NumberFormat	• It provides methods to determine the number formats of the locales and their name. It helps to format and parse numbers for any locales.

© Aptech Ltd. Web Component Development Using Java/Session 14 20

Formatting Currency 2-4

- ❖ Some of the methods of currency class are as follows:

public String getCurrencyCode()
--

- ❑ This method gets the currency code of the currency as per the ISO 4217 codes.
- ❑ The code snippet shows the use of the `getCurrencyCode()` method.

```
/* this snippet returns the currency code of the
   currency as per the ISO 4217 codes */
public String getCurrencyCode() {
    return currency.getCurrencyCode();
}
```

© Aptech Ltd. Web Component Development Using Java/Session 14 21

Formatting Currency 3-4

public String getSymbol()

- ❑ This method gets the currency symbol for the default locale.
- ❑ The code snippet shows the use of `getSymbol()` method.

```
/* this snippet gets the currency symbol for
   default locale */
public String getCurrencySymbol() {
    return currency.getSymbol();
}
```

© Aptech Ltd. Web Component Development Using Java/Session 14 22

Formatting Currency 4-4

```

public static Currency getInstance(Locale locale)
{
    // This method returns the Currency instance for the country of the specified locale.
    // The code snippet shows the use of getInstance() method.

    // this snippet gets the currency instance //
    import java.text.NumberFormat;
    import java.util.Currency;

    public class CurrencyClass {
        public static void main ( String args []){
            // returns locale-specific currency instance
            Currency localeCurrency =
            Currency.getInstance(locale);
        }
    }
}

```

© Aptech Ltd. Web Component Development Using Java/Session 14 23

Use slides 20 to 23 to explain formatting currency.

The class `Currency` represents currencies.

Tell the students that by using `NumberFormat` class, a user can format the currency according to the locale. The `getCurrencyInstance()` method of the `NumberFormat` class returns the instance of the `NumberFormat` class.

Explain the methods of the `Currency` class as mentioned on slides 21 and 22.

The following additional example shows how to format currency for a particular locale in the servlet:

```

import java.text.NumberFormat;
import java.util.Date;
import java.util.Locale;
. . .
    NumberFormat nft =
NumberFormat.getCurrencyInstance(Locale.KOREA);
    String formattedCurr = nft.format(1000000);
    out.println("Korea: " + formattedCurr);
. . .

```

The following additional code snippet shows the use of `Currency` class:

```

// Creates a currency object with specified locale
Locale locale = Locale.GERMANY;
Currency curr = Currency.getInstance(locale);

// Prints currency's code
System.out.println("Locale's currency code:" +
curr.getCurrencyCode());

```

Slides 24 to 28

Let us understand formatting numbers.

Formatting Numbers 1-5

- ❖ Some of the methods of NumberFormat class are as follows:

```
public final String format(double number)
```

- ❑ This method is the specialization of the format.
- ❑ The code snippet shows the use of `format()` method.

```
/* getPercentInstance() returns a percentage
format for the current default locale */
NumberFormat nft =
NumberFormat.getPercentInstance(locale);
String formatted = nft.format(0.51);
```

© Aptech Ltd. Web Component Development Using Java/Session 14 24

Formatting Numbers 2-5

```
public Currency getCurrency()
```

- ❑ This method provides the number format while formatting currency values.
- ❑ The value derived initially is locale dependent.
- ❑ If no valid currency is determined and no currency has been set using `setCurrency` the returned value may be null.
- ❑ The code snippet shows the use of `getCurrency()` method.

```
// This snippet display the currency as per the locale
import java.text.NumberFormat;
import java.util.Currency;

public class CurrencyClass{
    public static void main ( String args []){
        NumberFormat formatter = NumberFormat.getInstance () ;
        System.out.println ( formatter.getCurrency () );
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 14 25

Formatting Numbers 3-5

```
public static final NumberFormat getInstance()
```

- ❑ This method returns the default number format for the current default locale.
- ❑ The code snippet shows the use of `getInstance()` method.

```
// this snippet display the currency as per the locale //
import java.text.NumberFormat;
import java.util.Currency;

public class CurrencyClass{
    public static void main ( String args[]){
        NumberFormat formatter = NumberFormat.getInstance();
        System.out.println ( formatter.getCurrency());
    }
}
```

© Aptech Ltd. Web Component Development Using Java/Session 14 26

Formatting Numbers 4-5

```
public Number parse(String str) throws ParseException
```

- ❑ This method parses text from the beginning of the specified string to produce a number.
- ❑ The code snippet shows the use of `parse()` method.

```
NumberFormat nf = NumberFormat.getInstance();  
Number myDemoNumber = nf.parse(myDemoString);
```

© Aptech Ltd.

Web Component Development Using Java/Session 14

27

Formatting Numbers 5-5

```
public void setCurrency(Currency currency)
```

- ❑ This method sets the currency used by the number format when formatting currency values.
- ❑ This does not update the minimum or maximum number of fraction digits used by the number format.
- ❑ The code snippet shows the use of `setCurrency()` method.

```
// Sets the currency to new amountCurrency  
NumberFormat formatter =  
NumberFormat.getInstance();  
formatter.setCurrency(amountCurrency);
```

© Aptech Ltd.

Web Component Development Using Java/Session 14

28

Use slides 24 to 28 to explain how to format numbers.

Tell them that `NumberFormat` class provides the methods to format the numbers.

The following additional code snippet shows the formatting of numbers:

```
numberFormat ntf = NumberFormat.getNumberInstance(Locale.KOREA);  
ntf.setParseIntegerOnly(false);  
double usersNumber = 1976.0826;  
  
out.println("User's number: " + ntf.format(usersNumber));
```

Slides 29 and 30

Let us understand formatting percentages.

Formatting Percentages 1-2

- ❖ The format for displaying percentages can be changed using `getPercentInstance()` method of `NumberFormat` class.
- ❖ For example, with this format, a fraction as `0.82` can be displayed as `82%`.
- ❖ The two methods for formatting percentage are as follows:

<code>getPercentInstance()</code> <ul style="list-style-type: none"> • This method returns a percentage format for the current default locale. • Syntax: <code>public static final NumberFormat getPercentInstance()</code> 	<code>getPercentInstance(Locale inLocale)</code> <ul style="list-style-type: none"> • This method returns a percentage format for the specified locale. • Syntax: <code>public static NumberFormat getPercentInstance(Locale inLocale)</code>
---	---

© Aptech Ltd. Web Component Development Using Java/Session 14 29

Formatting Percentages 2-2

- ❖ Figure depicts the code for formatting of percentages.

```

static public void displayPercent(Locale currentLocale) {
    Double percent = new Double(0.75);
    NumberFormat percentFormatter;
    String percentOut;

    percentFormatter = NumberFormat.getPercentInstance(currentLocale);
    percentOut = percentFormatter.format(percent);
    System.out.println(percentOut + " " + currentLocale.toString());
}

```

Output:

75% en_US

© Aptech Ltd. Web Component Development Using Java/Session 14 30

Use slides 29 to 30 to explain formatting percentages.

Tell them that the format for displaying percentages can be changed using `getPercentInstance()` method of `NumberFormat` class.

The following code snippet shows an example for formatting percentages:

```

    .
    .
    .

    // Format percentage
    Locale locale = Locale.CHINA;
    String string =
    NumberFormat.getPercentInstance(locale).format(545.23);

}

}


```

OUTPUT 54,523%

Slides 31 and 32

Let us understand formatting messages.

Formatting Messages 1-2

- Formatting also helps provide messages in user's language.
- In order to format a message, the `MessageFormat` object is used.
- The array of objects using the format specifiers embedded in the pattern is formatted by `MessageFormat.format()` method.
- It returns the result as a `StringBuffer`.

© Aptech Ltd. Web Component Development Using Java/Session 14 31

Formatting Messages 2-2

- ❖ The classes for formatting the message are as follows:
 - MessageFormat**
 - This class provides a means to generate related messages in a language-neutral way.
 - `MessageFormat` takes a set of objects, formats them, then inserts the formatted strings into the pattern at the appropriate places.
 - It can also be used to construct messages displayed for end users.
 - MessageFormat.Field**
 - This class defines constants that are used as attribute keys in the `AttributedCharacterIterator` object returned from `MessageFormat.formatToCharacterIterator()`.
- ❖ Figure depicts message formatting.

© Aptech Ltd. Web Component Development Using Java/Session 14 32

Use slides 31 and 32 to explain formatting messages.

Tell them that Message formatting is required so that the end user is benefited. The formatting provides message in language neutral way. Formatting also helps provide messages in user's language. In order to format a message the `MessageFormat` object is used. The array of objects using the format specifiers embedded in the pattern is formatted by `MessageFormat.format()` method. It returns the result as a `StringBuffer`.

Tips:

The following additional example shows the use of MessageFormat:

...

```
String pattern = "The time is {0,time} and your number is {1,number}  
.";  
MessageFormat format = new MessageFormat(pattern);  
Object objects[] = { new Date(), new Integer((int) (Math.random()  
) * 1000) };  
String formattedOutput = format.format(objects);  
System.out.println(formattedOutput);  
...
```

Slides 33 to 40

Let us understand internationalizing JSP pages.

Internationalizing JSP Pages

- ❖ JSP Standard Tag Library (JSTL) provides a set of internationalization or I18N tags that are used for applying various internationalization formats.
- ❖ The tag library helps reduce and manage the complexities of internationalized applications.
- ❖ Several tags available in I18n tag library are as follows:
 - ❑ formatDate
 - ❑ formatNumber
 - ❑ message

© Aptech Ltd. Web Component Development Using Java/Session 14 33

Formatting Dates in JSP

- ❖ The tag <fmt:formatDate> is used for formatting dates and/or time in JSP for internationalization.
- ❖ The value attribute or the body content of the <fmt:formatDate> tag formats the date value.
- ❖ The formatted date is written to the JSP's writer.
- ❖ The value can also be stored in a string named var and an optional scope attribute.
- ❖ Syntax:


```
<fmt:formatDate value="date" [type="{time|date|both}"]  
[dateStyle="{default|short|medium|long|full}"]  
[timeStyle="{default|short|medium|long|full}"]  
[pattern="customPattern"]  
[TimeZone="TimeZone"]  
[var="varName"]  
[scope="{page|request|session|application}"]/>
```
- ❖ Example:<fmt:formatDate value="\${FinishDate}" type="date" dateStyle="full"/>

© Aptech Ltd. Web Component Development Using Java/Session 14 34

Formatting Currencies in JSP 1-3

- ❖ The currencies can be formatted in JSP for internationalization using JSTL I18N tags.
- ❖ The <fmt:setLocale> stores the specified locale in the javax.servlet.jsp.jstl.fmt.locale configuration variable.
- ❖ The formatting is done as per the set locale.
- ❖ The tag <fmt:formatNumber> can be used to format the currencies.
- ❖ The number is specified to be formatted either with an EL expression in value attribute or as the tag's body content.
- ❖ The desired formatting is specified by the type attribute.

<fmt:setLocale>

- ❖ Syntax:


```
<fmt:setLocale value="locale"  
[variant="variant"]  
[scope="{page|request|session|application}"]/>
```

© Aptech Ltd. Web Component Development Using Java/Session 14 35

Formatting Currencies in JSP 2-3

◆ Syntax:

```
<fmt:formatNumber>
<fmt:formatNumber value="numericValue"
[type="{number|currency|percent}"]
[pattern="customPattern"]
[currencyCode="currencyCode"]
[currencySymbol="currencySymbol"]
[groupingUsed="{true|false}"]
[maxIntegerDigits="maxIntegerDigits"]
[minIntegerDigits="minIntegerDigits"]
[maxFractionDigits="maxFractionDigits"]
[minFractionDigits="minFractionDigits"]
[var="varName"]
[scope="(page|request|session|application)"]/>
```

© Aptech Ltd. Web Component Development Using Java/Session 14 36

Formatting Currencies in JSP 3-3

◆ The code snippet demonstrates the formatting of currency in JSP.

```
//formatting for currency
<fmt:setLocale value="en_GB"/>
Formatting salary with Locale <B>en_GB</B> becomes :
<fmt:formatNumber type="currency" value="${salary}" /><BR>
```

© Aptech Ltd. Web Component Development Using Java/Session 14 37

Formatting Percentages in JSP

◆ The `<fmt:formatNumber>` tag formats a number in integer, decimal, currency, and percentage.

◆ By specifying the type attribute in `<fmt:formatNumber>` percentage of a number can be obtained.

◆ This happens when the value is multiplied with hundred.

Example: `<fmt:formatNumber value="0.82" type="percent"/>`

© Aptech Ltd. Web Component Development Using Java/Session 14 38

Formatting Messages in JSP 1-2

- ❖ The <fmt:message> tag retrieves a message from a resource bundle and optionally, uses the `java.util.MessageFormat` class to format the message.
- ❖ The key attribute specifies the message key.
- ❖ If the <fmt:message> tag occurs within a <fmt:bundle> tag, the key is appended to the bundle's prefix, if there is one.
- ❖ If the <fmt:message> tag occurs outside of a <fmt:setBundle> tag, the bundle attribute must be present and must be an expression that evaluates to a `LocalizationContext` object.
- ❖ A variable is initialized with the <fmt:setBundle> tag.

Syntax:

```
<fmt:message key="messageKey"
[bundle="resourceBundle"] [var="varName"]
[scope="(page|request|session|application)"]/>
```

© Aptech Ltd.

Web Component Development Using Java/Session 14

39

Formatting Messages in JSP 2-2

- ❖ The code snippet demonstrates the formatting of messages in JSP.

```
// This snippet formats the message
<fmt:message key="welcome">
    <fmt:param value="\${userNameString}"/>
</fmt:message>
```

- ❖ Figure depicts formatting of messages.



© Aptech Ltd.

Web Component Development Using Java/Session 14

40

Use slides 33 to 40 to explain how to internationalize JSP pages.

Use slide 34 to explain formatting dates in JSP. Tell the students that <fmt:formatDate> allows the user to format date and time based on a specific locale.

The attributes for <fmt:formatDate> method are as follows:

- Value - Date and/or time value to be formatted.
- Type - Accepts date or/and time to be used to format.
- dateStyle - Defines predefined formatted style for date only if date value is used for formatting.
- timeStyle - Defines predefined formatted style for time only if time value is used for formatting.
- Pattern - Defines standard customized pattern for formatting date and/or time.
- timeZone - Specifies time zone for that time value, if the time value is used to format.

- Var – Used for exporting scoped variable which stores the formatted date and/or time as a string.
- Scope - Defines the scope of var.

Use slides 35 to 37 to explain formatting currencies in JSP. Tell the students that formatNumber tag is also used to format currency and is done with the help of setLocale tag. setLocale tag has value attribute that consists of two parts, ISO language code and ISO country code.

Use slide 38 to explain formatting percentages in JSP. Tell the students that to display a number as percent, format Number tag is used.

Use slides 39 and 40 to explain formatting messages in JSP. Tell the students that message format tag is used to map the key to localized message.

The attributes of format message tag are as follows:

- key - provides the key to map with localized messages.
- bundle - provides localization context in whose resource bundle the message key is looked up.
- var - provides the name of the exported scoped variable which stores the localized message.
- scope - provides the scope of variable.

Slide 41

Let us summarize the session.

Summary

- ❖ Internationalization can be defined as the method of designing an application that can be adapted to a region or a language without much change in the technology.
- ❖ Localization is a process of making a product or service, language, culture, and local ‘look-and-feel’ specific.
- ❖ Internationalization of server-side code reduces the task of writing multiple versions of a class for different locales.
- ❖ Resource bundles are used to achieve the locale specific output.
- ❖ Internationalization requires formatting of dates, numbers, currencies, and messages with the help of resource bundle.
- ❖ The internationalization or I18N tags in JSTL help to reduce and manage the complexities of internationalized applications. There are several tags available in I18N.

© Aptech Ltd. Web Component Development Using Java/Session 14 41

In slide 41, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

14.3 Post Class Activities for Faculty

You should familiarize yourself with the topics of the next session. You should also understand how to secure Web applications.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the next session. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.

Session 15 – Securing Web Applications

15.1 Pre-Class Activities

Before you commence the session, you should familiarize yourself with the topics of this session in-depth. You should revisit topics of the previous session for a brief review.

Here, you can ask students the key topics they can recall from previous session. Prepare a question or two which will be a key point to relate the current session objectives.

15.1.1 Objectives

By the end of this session, the learners will be able to:

- Explain the need and features of securing Web applications
- Describe the HTTP basic, digest, client, and form-based authentication method of ensuring security
- Explain how to configure users in Tomcat
- Explain how to specify authentication mechanisms using web.xml
- Describe the seven steps to implement declarative security
- Explain the concept and five steps to implement programmatic security
- Describe the HttpServletRequest methods for identifying users

15.1.2 Teaching Skills

To teach this session successfully, you should be aware with the features of securing Web applications. Familiarize yourself with HTTP basic, digest, client, and form-based authentication method for ensuring security.

You should know how to specify authentication mechanisms using web.xml and also the steps to implement programmatic security. Also, aware yourself with HttpServletRequest methods for identifying users.

The faculty can refer the following link for more information on securing Web applications using NetBeans IDE - <https://netbeans.org/kb/docs/web/security-webapps.html>

For teaching in the class, you are expected to use slides and LCD projectors.

Tips:

It is recommended that you test the understanding of the students by asking questions in between the class.

In-Class Activities:

Follow the order given here during In-Class activities.

Overview of the Session:

Give the students a brief overview of the current session in the form of session objectives. Show the students slide 2 of the presentation.

Objectives

- ❖ Explain the need and features of securing Web applications
- ❖ Describe the HTTP basic, digest, client, and form-based authentication method of ensuring security
- ❖ Explain how to configure users in Tomcat
- ❖ Explain how to specify authentication mechanisms using web.xml
- ❖ Describe the seven steps to implement declarative security
- ❖ Explain the concept and five steps to implement programmatic security
- ❖ Describe the HttpServletRequest methods for identifying users

© Aptech Ltd. Web Component Development Using Java/Session 15 2

Tell the students that they will be introduced to the features of securing Web applications. In this session, they will learn HTTP basic, digest, client, and form-based authentication method for ensuring security.

The session also explains how to specify authentication mechanisms using `web.xml` and also the steps to implement programmatic security. Finally, the session explains `HttpServletRequest` methods for identifying users.

15.2 In-Class Explanations

Slides 3 and 4

Let us introduce three important issues in Internet security.

Introduction 1-2

- ❖ A Web application can be accessed using a Web browser over a network, such as Intranet or the Internet.
- ❖ Easier accessibility to Web applications opens door to attackers to access or modify the confidential information.
- ❖ It is necessary to secure the Web application to keep the information safe using some of the security mechanisms.
- ❖ Figure depicts unauthorized access on the Internet.

© Aptech Ltd. Web Component Development Using Java/Session 15 3

Introduction 2-2

- ❖ Three important issues to be considered in case of security:

Authentication

- Enables client verification through correct username and password.

Confidentiality

- Keeps information hidden from people other than the involved client and server.

Integrity

- Ensures that the content of the communication is not changed during transmission.

© Aptech Ltd. Web Component Development Using Java/Session 15 4

Use slides 3 and 4 to explain three important issues in Internet security.

A Web application is an application, which is accessed using a Web browser over a network, such as Intranet or the Internet. The reason of popularity of Web application is the ability to maintain the application and easy accessibility to information.

Easier accessibility to Web applications opens door to attackers to access or modify the confidential information. Hence, it is necessary to secure the Web application to keep the information safe using some of the security mechanisms.

Then, discuss with them about the issues that are to be considered in case of security:

- Authentication – This enables client verification through correct username and password.
- Confidentiality – This keeps information hidden from people other than the involved client and server.
- Integrity – This ensures that the content of the communication is not changed during transmission.

In-Class Question:

After you finish explaining three important issues in Internet security, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which issue keeps information hidden from people other than the involved client and server?

Answer:

Confidentiality

Slides 5 and 6

Let us understand Authentication and Authorization.

Understanding Authentication and Authorization 1-2

- ❖ When the user accesses the Web application resources, he/she needs to be identified.
- ❖ Identifying the user allows the Web application to understand:
 - Identity of the user
 - Operations that can be performed by the user
 - Maintaining integrity and confidentiality of the accessed resources

© Aptech Ltd. Web Component Development Using Java/Session 15 5

Understanding Authentication and Authorization 2-2

- ❖ Figure shows the various runtime security mechanism applied in a Web application.

```

graph TD
    CC[Client Call] --> TDP{Transport Data Protection}
    TDP -- Yes --> Encrypt[Encrypt]
    Encrypt --> Decrypt[Decrypt]
    Decrypt --> SR1[Service Request]
    TDP -- No --> SR1
    SR1 --> JWS[Java Web Server]
    JWS --> AU{Authenticate User}
    AU -- Fail --> RR1[Reject Request]
    AU -- Pass --> AUC{Authorized User Check}
    AUC -- Fail --> RR2[Reject Request]
    AUC -- Pass --> SR2[Service Request]
  
```

© Aptech Ltd. Web Component Development Using Java/Session 15 6

Use slides 5 and 6 to explain authentication and authorization.

Tell the students that authentication is the process of identifying the user and verifying his/her access to the Web application. Authorization is the process of allocating permissions to the authenticated users.

The user have to prove his/her identity to the server or client in authentication. In most cases, authentication by a server uses username and password. Authentication by a client requires the server to give a certificate to the client in which a trusted third party states that the server belongs to the entity that the client expects it to. Only when the user and the system are verified by authentication process, the client can access the Web application resources.

In authorization, server determines if the authenticated client has permission to use a resource or access a file.

Using the figure given on slide 6, explain the runtime security mechanism applied in a Web application.

Slide 7

Let us understand securing Web applications.

Securing Web Applications

- ❖ Web applications are secured by:
 - Application Developer - Set up the security for the application by using annotations or deployment descriptors.
 - Application Deployer - Assigns method permissions for security roles, set up authentication and transport mechanisms.
- ❖ Terms related to applying security in Java Web application are as follows:

User	Is an identity that has been defined in the Web server.
Group	Is a set of authorized users classified with common features such as job title or department.
Role	A role is a set of permissions that is applied to a resources in an application.
Realm	A realm is a database of users and groups identifying valid users for the Web applications.

© Aptech Ltd, Web Component Development Using Java/Session 15 7

Use slide 7 to explain securing Web applications.

Web applications are created by application developers who set up the security for the application by using annotations or deployment descriptors. This information is then used by the application deployers who assign method permissions for security roles, set up authentication, and transport mechanisms.

Then, explain the common terms used in applying security to Web applications.

Tell the students that a realm is a security policy domain defined for a Web or for an application server. The type of realms are as follows:

File realm – In this, the server stores user credentials locally in a file named keyfile.

Certificate realm - In this, the server stores user credentials in a certificate database.

Admin realm - In this, the server stores user administrator user credentials locally in a file named admin-keyfile.

Tips:

Some other terminologies is used to describe the security requirements are principal, security attributes, security domain, and credentials.

Slides 8 and 9

Let us understand configuring users in Tomcat.

Configuring Users in Tomcat 1-2

- ❖ For Tomcat 7, create a user with the manager-script role and a password for that users.
- ❖ Figure shows how to register the Tomcat server with NetBeans IDE.

© Aptech Ltd. Web Component Development Using Java/Session 15 8

Configuring Users in Tomcat 2-2

- ❖ The basic users and roles for the Tomcat server are in `tomcat-users.xml` which is available in:
 - `<CATALINA_BASE>\conf` directory under the installed directory of the machine.
- ❖ The code snippet demonstrates how to add users and roles in the `tomcat-users.xml` file.

```
// To include a new user
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="manager"/>
  <role rolename="admin"/>
  <user username="Tom" password="tempo"
        roles="admin,manager"/>
  <user username="admin" password="aptech"
        roles="admin,manager"/>
</tomcat-users>
```

© Aptech Ltd. Web Component Development Using Java/Session 15 9

Use slides 8 and 9 to explain configuring users in Tomcat server.

Explain the steps to configure Tomcat server in NetBeans IDE and then demonstrate on how to add users and roles in `tomcat-users.xml`.

Tips:

You can refer this link to know more about Tomcat server:

http://www.ntu.edu.sg/home/ehchua/programming/howto/Tomcat_More.html

Slides 10 to 12

Let us understand authentication mechanism.

Authentication Mechanism 1-3

- ❖ The security mechanisms are based on authentication and authorization techniques.
- ❖ Some of the commonly used authentication mechanisms are as follows:



© Aptech Ltd. Web Component Development Using Java/Session 15 10

Authentication Mechanism 2-3

HTTP Basic Authentication Method

- In HTTP basic authentication, the Web browser pops-up a login page, which prompts the user for credentials in the form of username and password.
- A user will be allowed to access information only if the credentials match a stored pair of username and password.

HTTP Digest Authentication Method

- In HTTP digest authentication, the server has the password based on a hash function.
- By using the hash function, the database stored the hash value rather than the password in plain text, so that it is difficult to decipher the data.

© Aptech Ltd. Web Component Development Using Java/Session 15 11

Authentication Mechanism 3-3

Form-based Authentication Method

- In form-based authentication, the browser provides a login form, which appears in response to a request.
- The user can enter a username and password in the form.
- If the entered credentials match a stored pair of username and password, the user will be allowed to access information.
- The user will be directed to an error page if the login fails

HTTPS Client Authentication Method

- In HTTPS client authentication, the browser uses HTTPS protocol, which is identical with the http but the URL indicates to use a default TCP port and an extra authentication layer in between HTTP and TCP.
- This extra security layer conforms the client's authentication.

© Aptech Ltd. Web Component Development Using Java/Session 15 12

Use slides 10 to 12 to explain the different types of authentication mechanism.

Slide 13

Let us understand HTTP basic authentication.

HTTP Basic Authentication

- ❖ HTTP basic authentication:
 - Web browser pops-up a login page in response to a request.
 - Login page prompts the user for credentials, such as username and password.
 - Works on the assumption that the client-server communication is reliable.
 - Does not provide any protection for the information communicated between the client and the server.
- ❖ The advantage of using basic authentication is that it is browser friendly.
- ❖ Figure depicts basic authentication.



© Aptech Ltd. Web Component Development Using Java/Session 15 13

Use slide 13 to explain HTTP basic authentication.

Tell the students that HTTP basic authentication requires a server request for username and password from the Web client. The verification of username and password are validated by comparing them against a database of authorized users in the specified realm.

Following actions takes place, when the basic authentication is declared:

- Access to a protected resource is requested by the client.
- A dialog box is returned by the Web server that requests the username and password.
- The client submits the username and password to the server.
- The server authenticates the user in the specified realm and returns the requested resource, if it is successful.

Tips:

Refer to this link for getting an example on how to configure basic authentication with a servlet:
<http://docs.oracle.com/cd/E19798-01/821-1841/bncck/index.html>

In-Class Question:

After you finish explaining HTTP basic authentication, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



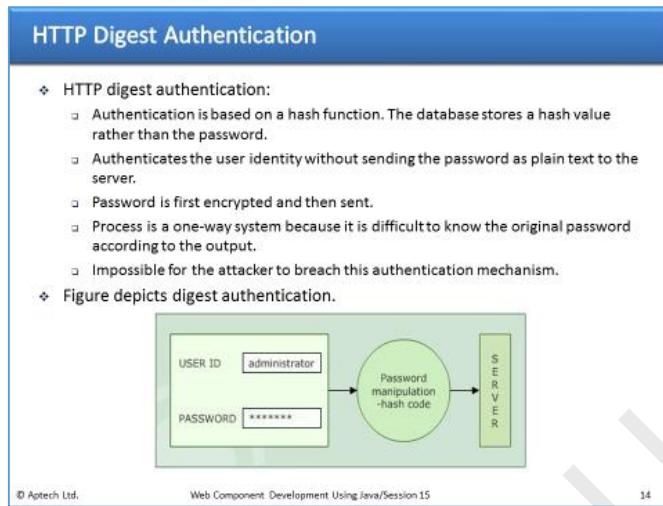
Which authentication requires a server request for username and password from the Web client?

Answer:

HTTP basic authentication

Slide 14

Let us understand HTTP digest authentication.



Use slide 14 to explain HTTP digest authentication.

Tell the students that HTTP digest authentication authenticates a user with a username and a password. It does not send user passwords over the network.

Advantages of HTTP digest authentication are as follows:

- The credentials are hashed.
- Replay attacks are prevented.
- To prevent reuse, the server creates a list of recently issued or used server nonce values.

The main disadvantage of HTTP digest authentication is the client could not verify the server's identity.

Tips:

To know more about Http Authentication, refer this link:

<http://docs.oracle.com/javase/7/docs/technotes/guides/net/http-auth.html>

Slide 15

Let us understand HTTPS client authentication.

HTTPS Client Authentication

- ❖ **HTTP client authentication:**
 - Is a secured client authentication technique based on Public Key Certificates.
 - Is similar to http but uses the https, which is HTTP over Secure Socket Layer (SSL).
 - Based on URL that instructs the browser to use a default TCP port with an extra authentication layer in between HTTP and TCP.
- ❖ Figure depicts HTTPS client authentication.

```
graph LR; CLIENT[CLIENT] -->|HTTP Security Layer-1| HTTPS[HTTPS]; HTTPS -->|HTTPS Security Layer-2| SERVER[SERVER]; SERVER -->|HTTPS Security Layer-2| HTTPS; HTTPS -->|HTTP Security Layer-1| CLIENT;
```

© Aptech Ltd, Web Component Development Using Java/Session 15 15

Use slide 15 to explain the HTTPS client authentication. Tell the students that HTTPS client authentication requires the client to hold a Public Key Certificate (PKC). It is more secure than other methods of authentications.

In-Class Question:

After you finish explaining HTTPS client authentication, you will ask the students an In-Class question. This will help you in reviewing their understanding of the topic.



Which authentication requires the client to hold a Public Key Certificate (PKC)?

Answer:

HTTPS client authentication

Slide 16

Let us understand form-based authentication.

Form-based Authentication

- ❖ Form-based authentication:
 - Provides a login page, which appears in response to a request from the user.
 - An error page is also available if the login fails from the user side.
 - After getting the username and password validated by the server, the access is provided to the user.
 - Here, a user can define how to protect the Web content for the URLs you are planning to protect.
- ❖ Figure depicts form-based authentication.

```

graph TD
    A[LOGIN FORM] --> B{Authentication  
YES/NO}
    B -- YES --> C[Home Page]
    B -- NO --> D[Error Page]
  
```

© Aptech Ltd. Web Component Development Using Java/Session 15 16

Use slide 16 to explain form-based authentication. Tell the students that form-based authentication allows the developer to control the login authentication screens by customizing the login screen and error pages that an HTTP browser presents to the end user.

The following actions takes place, when the form-based authentication is declared:

- A request to access a protected resource is sent by the client.
- The server redirects the client to a login page, if the client is unauthenticated.
- The login form to the server is submitted by the client.
- The server attempts to authenticate the user.
- The server redirects the client to the resource using the stored URL path, if the user is authorized.
- The client is redirected to an error page, if authentication fails.

Tips:

To create a working example based on form-based authentication with a servlet, refer this link:

<http://docs.oracle.com/cd/E19226-01/820-7627/bncby/index.html>

Slides 17 and 18

Let us understand configuring authentication in web.xml.

Configuring Authentication in web.xml 1-2

- The sub elements of `<login-config>` element are as follows:

<code><auth-method></code>	This sub element names the authentication method used in the element.
<code><realm-name></code>	This sub element names the Web resource where the <code><login-config></code> maps.
<code><form-login-config></code>	This sub element is optional, and is specified only when using form-based authentication that is the <code><auth-method></code> value is set to FORM.
<code><form-login-page></code>	This sub element specifies the form to display when a request is made to a protected Web resource in the Web application.

© Aptech Ltd. Web Component Development Using Java/Session 15 17

Configuring Authentication in web.xml 2-2

- The code snippet uses the form-based authentication mechanism.

```
<web-app>
  ...
  <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/login.htm</form-login-page>
      <form-error-page>/loginerror.htm</form-error-page>
    </form-login-config>
  ...
</web-app>
```

© Aptech Ltd. Web Component Development Using Java/Session 15 18

Use slides 17 and 18 to explain configuring authentication in web.xml.

Slide 19

Let us understand authorization strategies.

Authorization Strategies

- ❖ The Java security model has two strategies for providing access control namely:
 - Declarative access control
 - Programmatic access control

© Aptech Ltd. Web Component Development Using Java/Session 15 19

Use slide 19 to explain authorization strategies.

The main purpose of the security model is to control access to business services and resources in the application. The Java security model supports two strategies for providing access control namely declarative access control and programmatic access control.

Slides 20 to 26

Let us understand implementing declarative security.

Implementing Declarative Security 1-7

- ❖ The declarative security provides security to a resource with the help of the server configuration.
- ❖ Declarative security works as a different layer from the Web component with which it works.

Advantages of Declarative Security	Disadvantages of Declarative Security
<ul style="list-style-type: none"> • It gives scope to the programmers to ignore the constraints of the programming environment. • Updating the mechanism does not require total change in security model. • It is easily maintainable. 	<ul style="list-style-type: none"> • Access is provided to all or denied. • Access is provided by the server only if the password matches. • It cannot use both form-based and basic authentication for the same page.

© Aptech Ltd. Web Component Development Using Java/Session 15 20

Implementing Declarative Security 2-7

- ❖ To implement the declarative security, perform the following eight steps:

- Set up usernames, passwords, and roles
 - When a user tries to access a secured resource in a Web application, which uses form-based authentication, the system uses a form to ask for a username and a password.
 - After verifying the password, the system gives access to the user.
 - Then, it determines the role such as user, administrator or executive, which is defined by the user.
- Tell the Server that you are using Form-based authentication and designate locations of login and login failure pages
 - After validating the user and the role of user, it uses a form-based authentication, which supplies a value for the auth-method sub element, and uses the form-login-config sub element to give the locations of the login and login-failure pages.
 - These pages can consist of either JSP or HTML.

© Aptech Ltd. Web Component Development Using Java/Session 15 21

Implementing Declarative Security 3-7

- Create a Login page
 - The login-config element informs the server to use form-based authentication for creating a login page.
 - The login page requires a form for security check, a text field named username and a password field named password.
 - Redirect unauthenticated users to a designated error page.
 - The code snippet indicates that the container is using form-based authentication.

```

<FORM ACTION="securityServlet" METHOD="POST">
<TABLE>
<TR><TD>User name: <INPUT TYPE="TEXT" NAME="j_username">
<TR><TD>Password: <INPUT TYPE="PASSWORD" NAME="j_password">
<TR><TH><INPUT TYPE="SUBMIT" VALUE="Log In">
</TABLE>
</FORM>
  ...
  
```

© Aptech Ltd. Web Component Development Using Java/Session 15 22

Implementing Declarative Security 4-7

- **Create a page to report failed login attempts**
 - The `login-failure` page contains a link to an unrestricted section of the Web application.
 - The link shows 'login failed' or 'username and password not found' message.
- **Specifying URLs that should be password protected**
 - Specifying the URLs and describing the protection they should have, are the purposes of the `security-constraint` element.
 - The `security-constraint` element should come before the `login-config` element in `web.xml`, and contains four sub elements, `display-name`, `web-resource-collection`, `auth-constraint`, and `user-data-constraint`.

© Aptech Ltd.

Web Component Development Using Java/Session 15

23

Implementing Declarative Security 5-7

- The code snippet shows the use of `security-constraint`.

```

    ...
<web-app>
<!-- ... -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Sensitive</web-resource-
name>
    <url-pattern>/sensitive/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>administrator</role-name>
    <role-name>executive</role-name>
  </auth-constraint>
</security-constraint>
<login-config>...</login-config>
</web-app>

```

© Aptech Ltd.

Web Component Development Using Java/Session 15

24

Implementing Declarative Security 6-7

- **Specifying URLs that should be available only with SSL**
 - If your server supports SSL, you can define that certain resources are available only through encrypted HTTPS (SSL) connections.
 - Use of SSL does not obstruct the basic way that form-based authentication works.
 - The `user-data-constraint` sub element of `security-constraint` can mandate that certain resources be accessed only with SSL.
 - The code snippet instructs the server to permit only https connections to the associated resource.

```

<security-constraint>
<!-- ... -->
<user-data-constraint>
  <transport-guarantee>CONFIDENTIAL</transport-
guarantee>
</user-data-constraint>
</security-constraint>

```

© Aptech Ltd.

Web Component Development Using Java/Session 15

25

Implementing Declarative Security 7-7

- Turning off these invoker servlet
 - The most portable approach of turning off the invoker servlet is to remap the servlet pattern in the Web application, so that all requests that include the pattern are sent to the same servlet.
 - To remap the pattern, first create a simple servlet that prints an error message.
 - Then, use the `servlet` and `servlet-mapping` elements to send requests that include the servlet pattern to that servlet.
- Write authentication filter
 - Write the Servlet filter and mention in `web.xml`. This `Filter` class filters the HTTP requests based on its type.

© Aptech Ltd.

Web Component Development Using Java/Session 15

26

Use slides 20 to 26 to explain implementing declarative security.

Tell the students that there are mainly three ways to implement declarative security.

They are as follows:

- Security providers via the Administration Console.
- Java Authorization Contract for Containers (JACC).
- Deployment descriptor

Declarative security shows an application component's security requirements by using deployment descriptors. Deployment descriptors includes information that specifies how security roles and access requirements are mapped into environment-specific security roles, users, and policies.

Then, explain the steps to implement the declarative security as given on slides 21 to 26.

Tips:

You can refer this link for more information on declarative security:

<http://www.informit.com/articles/article.aspx?p=26139>

Slide 27

Let us understand programmatic security.

Programmatic Security

- ❖ Programmatic security authenticates the users and grants access to the users.
- ❖ The servlet or JSP page either authenticates the user or verifies that the user has authenticated earlier, to check the unauthorized access.
- ❖ To ensure the safety of network data, the servlet or JSP page redirects the HTTP connection to HTTPS of the URLs.

Advantages of Programmatic Security	Disadvantages of Programmatic Security
Ensures total portability	Much harder to code and maintain
Allows password matching strategies	Every resource must use the code

© Aptech Ltd. Web Component Development Using Java/Session 15 27

Use slide 27 to explain that programmatic security in an application is used to make security decisions. Programmatic security is effective when declarative security alone is not sufficient to show the security model of an application.

Slides 28 to 31

Let us understand HttpServletRequest methods for identifying user.

'HttpServletRequest' Methods for Identifying User 1-4

- ❖ The methods in the HttpServletRequest interface used for identifying a user by the help of cookies are as follows:

getAuthType()

This method returns the authentication scheme name.

Syntax:

```
public java.lang.String getAuthType()
```

getCookies()

It returns an array, which have all the information of the Cookie objects sent by the client.

Syntax:

```
public Cookie[] getCookies()
```

© Aptech Ltd. Web Component Development Using Java/Session 15 28

'HttpServletRequest' Methods for Identifying User 2-4

getHeader()

It returns the value of the header as a String, which is requested.

Syntax:

```
public java.lang.String getHeader(java.lang.String name)
```

getRemoteUser()

If the user is authenticated, it returns the login name of the user, else it returns null.

Syntax:

```
public java.lang.String getRemoteUser()
```

© Aptech Ltd. Web Component Development Using Java/Session 15 29

'HttpServletRequest' Methods for Identifying User 3-4

getRequestedSessionId()

This method returns the session ID that is defined by the client.

Syntax:

```
public java.lang.String getRequestedSessionId()
```

getSession()

This method returns the current session or creates one if there is no session.

Syntax:

```
public HttpSession getSession()
```

© Aptech Ltd. Web Component Development Using Java/Session 15 30

'HttpServletRequest' Methods for Identifying User 4-4

- isUserInRole()**
 - ❑ It returns a boolean value, which indicates whether the authenticated user is included in the logical 'role'.
 - ❑ **Syntax:**

```
public boolean  
isUserInRole(java.lang.String role)
```
- getUserPrincipal()**
 - ❑ It checks the validity of the requested session ID.
 - ❑ **Syntax:**

```
public java.security.Principal  
getUserPrincipal()
```
- isRequestedSessionIdValid()**
 - ❑ This method returns a `java.security.Principal` object.
 - ❑ **Syntax:**

```
public boolean  
isRequestedSessionIdValid()
```

© Aptech Ltd. Web Component Development Using Java/Session 15 31

Use slides 28 to 31 to explain HttpServletRequest methods for identifying users.

Slides 32 and 33

Let us understand implementing programmatic security.

Implementing Programmatic Security 1-2

- ❖ The six steps to implement programmatic security are as follows:
 - Check whether there is an authorization request header**
 - It checks whether the header exists or not.
 - Get the string, which contains the encoded username/password**
 - If the authentication header exists it returns the string, which contains the username and password in base64 encoding format.
 - Reverse the base64 encoding of the username/password string**
 - Reverse the base64 encoding of username or password by using the decodeBuffer method of BASE64Decoder class.
 - Check the username and password**
 - The common approach of checking the username and password is to use database to get the original username and password.
 - In other way, put the information of the password in the servlet.
 - You will get access if the incoming username and password matches the reference username and password pairs.

© Aptech Ltd. Web Component Development Using Java/Session 15 32

Implementing Programmatic Security 2-2

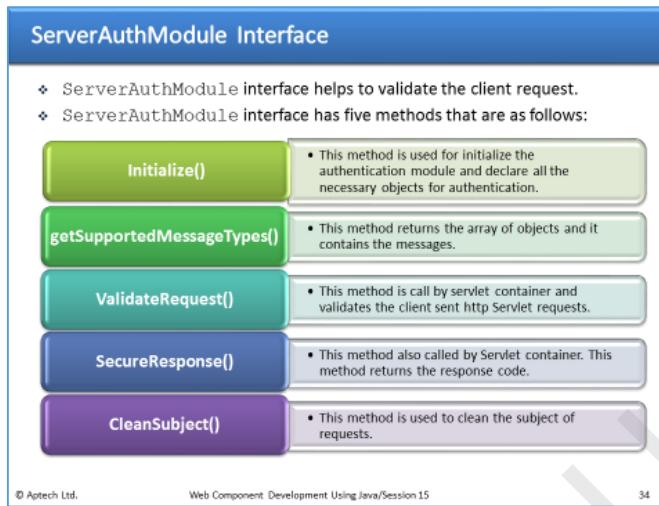
- If authentication fails, send the proper response to the client**
 - It returns a 401 response code, which is an unauthorized one in the form of, WWW-Authenticate: BASIC realm="some-name".
 - This response pops-up a dialog box requesting the user to enter a name and password for some-name.
- Servlet Annotations**
 - Based on J2EE API, javax.servlet.annotation package provides the number of classes and interfaces.
 - Using of these classes and interfaces declare the filters, servlets, and listeners to authenticate the http requests.

© Aptech Ltd. Web Component Development Using Java/Session 15 33

Use slides 32 and 33 to explain about implementing programmatic security.

Slide 34

Let us understand ServerAuthModule Interface.



Use slide 34 to explain ServerAuthModule Interface.

Tell the students that ServerAuthModule validates client requests and secures responses to the client.

Explain that a module implementation should be used to secure different requests as different clients. A module should also be used concurrently by multiple callers. The main responsibility of module implementation is to properly save and restore any state as necessary.

Slide 35

Let us summarize the session.

Summary

- ❖ A Web application is an application, which is accessed with the help of a Web browser.
- ❖ The reason of popularity of Web application is the ability to maintain it without changing the client computers. When you access the Web, hackers can get your information. So to keep your information secret, it is necessary to secure Web applications.
- ❖ There are four authentication mechanisms available. These are HTTP Basic Authentication, HTTP Digest Authentication, HTTPS Client Authentication, and Form-Based Authentication.
- ❖ To configure a user in Tomcat, first include the Tomcat 6.0 from Apache Software Foundation. When browser loads a resource, which is secured by web.xml file, the browser responds in two ways.
- ❖ The browser challenges the user if you are using basic authentication or forwards the login page if you are using form-based authentication.
- ❖ The declarative security provides security to a resource with the help of server configuration.
- ❖ Programmatic security authenticates the users and grants access to the users.

© Aptech Ltd. Web Component Development Using Java/Session 15 35

In slide 35, you will summarize the session. You will end the session, with a brief summary of what has been taught in the session.

15.3 Post Class Activities for Faculty

This session ends the **Web Component Development Using Java** course. Ask the students some questions related from all the topics which will help you to know the learnings taken by the students. You can solve the queries related to other sessions taught in the course.

Tips:

You can also check the Articles/Blogs/Expert Videos uploaded on the OnlineVarsity site to gain additional information related to the topics covered in the course. You can also connect to online tutors on the OnlineVarsity site to ask queries related to the sessions.