20IT605/JO01

Hall Ticket Number:

III/IV B.Tech (Regular) DEGREE EXAMINATION

July/August, 2023 — **Information Technology**

Sixth Semester — **Enterprise Programming**

Time: Three Hours — **Maximum:** 70 Marks

Answer question 1 compulsory. **(14X1 = 14Marks)**

Answer one question from each unit. **(4X14=56 Marks)**

			CO	BL	M
1	a)	What is the difference between web server and application server? A web server serves web content like HTML, while an application server processes dynamic logic and database interactions for applications.	CO1	L2	1M
	b)	Which JDBC driver is fastest and used more commonly? Thin driver is the fastest and most commonly used driver among all 4 jdbc drivers	CO1	L2	1M
	c)	Explain the meaning of the dynamic web page? A dynamic web page is a web page that can change its content or appearance based on user interactions or data from a database, providing personalized and interactive experiences.	CO1	L2	1M
	d)	Explain the JSP while loop. In JSP, a while loop is a control structure that repeatedly executes a block of code as long as a specified condition is true. The condition is checked before each iteration, and if it evaluates to true, the loop continues; otherwise, it exits.	CO2	L2	1M
	e)	Describe the navigation model of JavaServer Faces technology? JavaServer Faces (JSF) navigation involves rules in faces-config.xml, guiding how users move between views based on actions like button clicks, enhancing user interaction.	CO2	L2	1M
	f)	Describe the user interface component model of JavaServer Faces technology? JSF's UI component model uses reusable components for visual elements, fostering modular page creation and code reuse.	CO2	L2	1M
	g)	What are the key annotations provided in the JAX-RS API? In the JAX-RS API, key annotations include: 1. @Path: 2. @GET, @POST, @PUT, @DELETE: @PathParam. 3. @QueryParam. 4. @Consumes: 5. @Produces: 6. @HeaderParam: 7. @FormParam 8. @BeanParam: 9. @Context:	CO3	L2	1M
	h)	How do you set message headers in client request using JAX-RS API? Use the header() method with Invocation.Builder to set message headers in a JAX-RS client request.	CO3	L3	1M
	i)	How do you set filters in client request or response using JAX-RS API? Implement `ClientRequestFilter` to set filters for client requests and `ClientResponseFilter` for client responses using the JAX-RS API. Register them with the `Client` instance to apply the filters.	CO3	L3	1M

	j)	Define an enterprise bean. An enterprise bean is a server-side Java EE component that encapsulates business logic and is managed by the Java EE container for services like lifecycle management and transactions.	CO4	L3	1M
	k)	List out some annotations in session bean. @stateless @stateful @Local @Remote @post construct @predestroy @EJB @resource	CO4	L3	1M
	l)	Why use WebSocket over HTTP? WebSocket is preferred over HTTP for real-time communication, reduced latency, efficient data transfer, bi-directional interaction, push notifications, and lower overhead.	CO4	L2	1M
	m)	What is a transaction in EJB? In EJB, a transaction is a unit of work involving multiple related operations that are either completed together or rolled back in case of errors, ensuring data consistency and integrity.	CO4	L2	1M
	n)	Is threading is possible in EJB? Yes, threading is possible in EJB, but the EJB container manages threading to ensure thread safety and abstraction from low-level details.	CO4	L2	1M

Unit-I

2	a)	How do you package and deploy a Java EE Application? **Diagrams 3M and Explaination 4M**	CO1	L2	7M

Packaging and Deploying a JEE Application:

- Before applications can be deployed, they must be packaged into JAR (Java Archive), WAR (Web Archive) or EAR (Enterprise Archive) files.
- "Deployment" is the process that allows the application to work on the desired device.
- The packaged application includes "Deployment descriptors", which gives the information to the Application Server Software. That information includes mapping a URL to the application and connect it to the resources.
- When a request comes from the client, web server uses deployment descriptor file to map the URL of client's request to the specific component/module to handle it.
- A deployment descriptor describes how a component or a module should be deployed.
- The syntax of deployment descriptor files follows XML.
- NetBeans allows us to create, package and deploy JEE applications. It integrates the development environment with the deployment environment.
- Once we register the application server with the NetBeans IDE, we can easily deploy the applications, make changes and redeploy them.

1. WAR File:

- The file which is used for deploying the web components in a JEE application is called “Web Archive file” (WAR).
- WAR format is similar to ZIP file. It has a predefined structure consisting of a root directory (/META-INF) to hold HTML pages or JSP files or JSF files. ➢ The following example shows the structure of WAR file:

```
/META-INF
      /MANIFEST.MF
/WEB-INF
      /classes
            /javaeems

                  /chapter1
                        /web
                            /DisplayServlet.class
                            /WriteServlet.class
```

- This file has a special directory (/WEB-INF) under which it contains Java class files, Java servlets and other configuration information.
- This file has an extension (i.e) “.war”.

2. JAR File:

- The Enterprise Bean JAR uses /META-INF directory to store the configuration information about the “Enterprise Beans” together with the “java class files”.
- The following example shows the structure of JAR file:

```
/META-INF
      /MANIFEST.MF
      /persistence.xml
/javaeems
      /chapter1/
            /model
                  /ModelEJB.class
                  /Message.class
                  /MessageException.class
```

- This file has an extension (i.e) “.jar”.

3. EAR File:

- Some JEE applications need more configuration information. That information is present in a special configuration files called “deployment descriptor files”.
- These files are packaged in the /WEB-INF directory of WAR file and in the /META-INF of JAR file.
- The following example shows the structure of EAR file:

```
/META-INF
      /MANIFEST.MF
/HelloJavaEE-jar.jar
/HelloJavaEE-war.war
```

- This file has an extension (i.e) ".ear".
- It combines both the archives together along with the additional configuration information. The additional configuration information will be hold in /META-INF directory.

b) What is an enterprise Application? Explain architecture of an enterprise application. CO1 L2 7M

An enterprise application is a software application that is designed to meet the needs of a large organization. EAs are typically complex and scalable, and they are used to automate and manage core business processes.

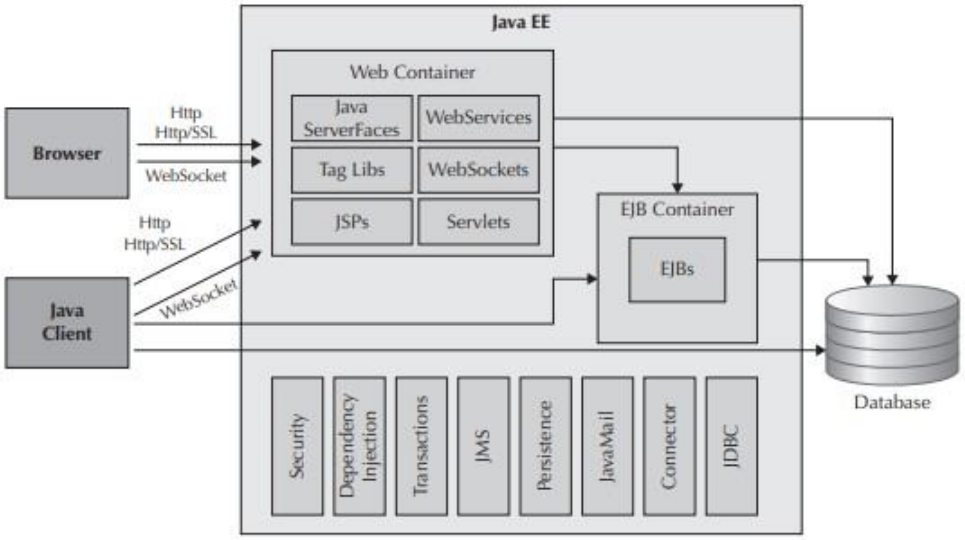


FIGURE 1-1. *Architecture of the Java EE platform*

• It is also known as

"Three-tier

Architecture"

(i.e): - Client-tier:

▪ It acts as a front end for an application.

▪ It consists of programs that interacts with the user.

▪ Those programs act as a user interface for applications and get the input from the user and then convert into request and forwarded to the middle tier.

▪ The server processes the request and returns the result and data to the client program.

▪ The client program translates the server response into text and presented

to the user.

▪ It holds the logic about how the application is interacted with the user.

▪ Through this tier, client can make a request to the middle tier.

- Server-tier:

▪ It consists of various containers for processing the request.

▪ Web container is used to deploy web applications.

▪ EJB container is used to deploy business applications.

▪ It holds the main logic about how the entire application works.

▪ It handles the client requests, process it and generate the response.

- Database-tier or EIS:

▪ It acts as a backend for an application.

▪ It holds the logic about how the data should be stored, fetched and displayed to the user.

• As a Java EE developer we have to mainly focussed on the middle tier to make the application easier, robust and secure.

• The large box labelled “Java EE” represents “Java EE Platform”. It refers to the runtime environment provided by the Java EE application server.

• All the code that we develop runs in that environment.

• That environment is also called as “Java EE Container”. It is made up of 2 other containers: a. Web container:

- To run the web components in a Java EE application.

- The web components are web pages, Java servlets, Web services, etc.

- Different web components can interact with the clients with standard web protocols. b. Enterprise JavaBeans container:

- To run the application logic part of a Java EE application.

- EJBs are java classes that contain and manipulate the core data structures of a Java EE application.

• The database tier of Java EE application holds all application data.

• The Java EE platform supports a variety of protocols that clients may interact with a Java EE application.

• The browser client connects to the Java EE application using standard web protocols such as HTTP and Web sockets.

• The small boxes in the Java EE container represents a variety of services that the application may choose to use.

(OR)

3 a) Write note on javax.servlet package and Servlet config. CO1 L3 7M

The **javax.servlet** package in Java is a part of the Java Servlet API, which provides a framework for building web applications. Servlets are Java classes that help in creating dynamic web content and handling requests and responses from web clients, such as web browsers. The **javax.servlet** package contains classes and interfaces that are essential for developing servlets and interacting with the servlet container, which is responsible for managing and executing servlets within a web server environment.

Key classes and interfaces in the **javax.servlet** package include:

1. Servlet: This interface defines the methods that servlets need to implement. It contains methods like **`init`, `service`, and `destroy`**, which are called by the servlet container during the lifecycle of a servlet.

2. Generic Servlet: This abstract class provides a base implementation of the `Servlet` interface. It's commonly extended to create servlets. It also extends the **ServletConfig** interface.

3. ServletConfig: This interface provides configuration information to a servlet. It allows a servlet to access initialization parameters defined in the deployment descriptor (web.xml) and provides methods to retrieve a servlet's name and reference to the servlet context.

4. ServletContext: This interface represents the context of a web application and provides methods for managing application-level resources, such as attributes, parameters, and more. It also offers methods for interacting with other web components within the application.

5. HttpServletRequest and HttpServletResponse: These classes represent the request and response objects exchanged between the web client and the servlet. They provide methods to access client data and construct server responses.

The **ServletConfig** is a crucial part of the servlet lifecycle and plays a role in configuring and initializing servlets. It's associated with each individual servlet and provides access to initialization parameters that are defined in the web.xml deployment descriptor. When a servlet is first loaded, its **`init`** method is called, and the **ServletConfig** object containing the initialization parameters is passed as an argument. This allows the servlet to retrieve configuration values specific to its needs.

In summary, the **`javax.servlet`** package is a fundamental part of the Java Servlet API, enabling the development of web applications. The **ServletConfig** object, provided by this package, facilitates the configuration and initialization of servlets by granting access to initialization parameters defined in the deployment descriptor.

b) Explain Http Servlet Request and Servlet Response interface. CO1 L3 7M

The `HttpServletRequest` and `HttpServletResponse` interfaces are essential components of the Java Servlet API, used for communication between a servlet and the client's web browser. They provide a way for servlets to receive requests from clients and send responses back. Let's delve into each interface's details:

1. **HttpServletRequest:**

The `HttpServletRequest` interface represents the information sent by the client's browser to the server as part of an HTTP request. It provides methods for retrieving various aspects of the client's request, including:

- **HTTP Method:** You can retrieve the HTTP method used in the request, such as GET, POST, PUT, DELETE, etc.
- **Request URI and URL:** You can access the URI (Uniform Resource Identifier) and URL (Uniform Resource Locator) of the requested resource.
- **Query Parameters:** If the request includes query parameters (e.g., in a URL like `example.com/page?param=value`), you can retrieve these parameters.
- **Headers:** You can access the headers sent by the client, such as User-Agent, Content-Type, etc.
- **Cookies:** If the client sent cookies, you can retrieve them using methods provided by the `HttpServletRequest` interface.
- **Session Management:** You can interact with the session associated with the request, which allows you to manage user-specific data across multiple requests.

2. **HttpServletResponse:**

The `HttpServletResponse` interface represents the response sent by the server to the client's browser. It provides methods for setting various aspects of the response, including:

- **Response Status:** You can set the HTTP status code to indicate the outcome of the request (e.g., 200 for OK, 404 for Not Found, 500 for Internal Server Error, etc.).
- **Headers:** You can add custom headers to the response, specifying attributes like Content-Type, Cache-Control, etc.
- **Cookies:** You can set cookies to be sent back to the client's browser.
- **Character Encoding and Content Type:** You can set the character encoding and content type of the response's body.
- **Response Output:** You can obtain a writer or an output stream to send data back to the client's browser.

Here's a simple example of how these interfaces are used within a servlet:

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

```
public class MyServlet extends HttpServlet {
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException {
       // Handling the request
       String paramName = request.getParameter("param");
       String userAgent = request.getHeader("User-Agent");

       // Creating the response
       response.setStatus(HttpServletResponse.SC_OK);
       response.setContentType("text/html");

       response.getWriter().println("<html><body>");
       response.getWriter().println("Parameter value: " + paramName + "<br>");
       response.getWriter().println("User-Agent: " + userAgent);
       response.getWriter().println("</body></html>");
   }
}
```

In this example, the servlet receives a parameter from the client's request using `HttpServletRequest` and sends a response with information back using `HttpServletResponse`.

In summary, `HttpServletRequest` and `HttpServletResponse` are crucial interfaces in the Java Servlet API, enabling servlets to interact with clients by processing incoming requests and crafting appropriate responses.

Unit-II

4 a) Explain the methodologies that can be implemented to create dynamic web pages. Discuss architecture of any one? CO2 L3 7M

The methodologies that can be implemented to create dynamic web pages.

• Java servlets are very powerful, flexible and provides a great foundation for web

applications.

• JEE provides a “Web component model” (JSP, JSF) to create/generate the

dynamic web content for all kinds of web browsers.

a. JSP:

- On the basis of Java servlet, the 1st variation of JEE is “Java Server Pages”.

- JSP code looks pretty much similar to HTML code.

- JSP tags are used to insert java code in the web page.

- It is a web-based technology helps us to create “Dynamic & Platform

independent web pages”.

- It is a type of Java servlet designed to fulfil the role of a user interface for a

java web application.

- JSP tags can be used to retrieve information from a database, registering user

preferences, accessing JavaBeans components.

- The embedded code snippet can become a reusable java component called as

"JSP tag library". It can be used by other JSPs.

b. JSF:

- It is the 2nd type of web component which provides a high-level framework to simplify the development of web-based user interfaces for server-side applications.

- It includes predefined user components like buttons, lists, etc together with the ability to validate the input and define user interaction flows.

• JEE platform contains a variety of web components that can handle various web services. Web services are categorized into 2 types (i.e) SOAP and REST.

• SOAP (Simple Object Access Protocol) is an XML-based protocol for accessing web services. It is a W3C recommendation for communication between 2 applications. It consumes more bandwidth and resources. These APIs uses "Web Services Description Language" for describing the functionalities being offered.

• REST (Representational State Transfer) is an architectural style not a protocol. These web services are fast, will consume less bandwidth and resources. These APIs uses "Web Application Description Language" for describing the functionalities being offered.

Introduction:

- JSP stands for "Java Server Pages".
- Java servlets supports the creation of dynamic web content.
- JEE provides a "Web component model" (JSP, JSF) to create/generate the dynamic

web content for all kinds of web browsers.

- JSP is the extension of "Servlet Technology". It provides more functionality than servlets.
- •Servlets provides a great foundation for JSP.
- It works similar to a servlet.
- When a servlet receives a client request, it will be processed and a response (in HTML)

will be prepared and sent back to the client.

- While generating the response HTML tags are embedded in method calling statements.
- Through servlets it is difficult to design complicated webpages as a response to the client.
- Implementation of JSP is simple and easy to maintain.

- JSP code looks pretty much similar to HTML code.
- A JSP page can include HTML, CSS, JavaScript & Java.
- The following are the various disadvantages of servlets:

1. Static html content is generated by a servlet.
2. Designing of response is difficult.
3. For every request we have to write service () overridden method inside each servlet.
4. Each of the client request will be handled by anyone of the servlets.
5. Whenever modification is made in the static content (presentation logic), then a servlet needs to recompiled and redeployed.

- In JSP, in order to process the client requests, we don't need to write the service ( ) overridden method.
- In JSP, in order to process the client requests, we can embed the java code through a
- "scriptlet tag" (<% code %>). Once a JSP page is created, we can send another request through it without overriding

service ( ) method.

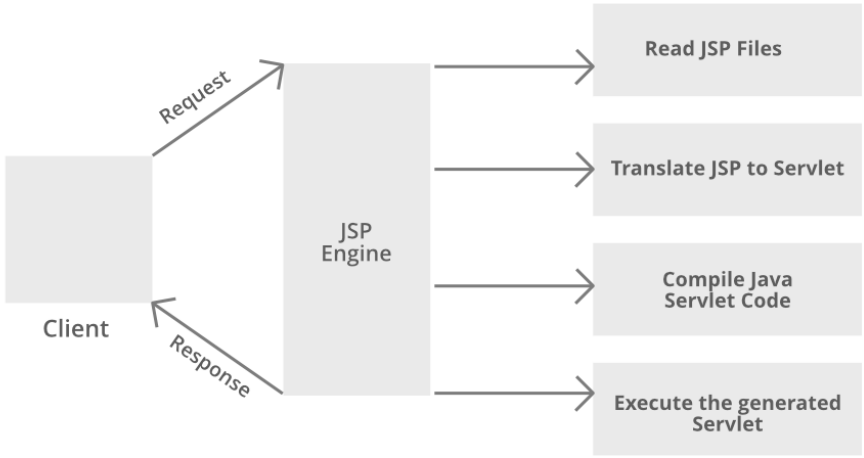
1. JSP offers JSTL library which provides both predefined tags and custom tags.

How JSP is translated into Servlets?

• We can't run JSP. We can only write code in JSP.
• Inside a web server we can run only servlets, not JSPs.
• The JSP code gets converted into servlets.
• The only advantage of writing JSP is, it is easier for the developer to write JSP files.
• If our intention is to create a page, JSP is better.
• If our intention is just to process it, Servlets are better.

JSP Runtime Architecture:

• JSP never really deals the HTTP request or the HTTP response.
• Once we deploy the JSP to the JEE web container, the web container reads the deployed JSP.
• From that information it will generate the java code for a servlet that fulfils the objectives of JSP.



JSP Scripting Elements:

1. Declarative Tag:

<%! Var's or Methods %>

Eg:

```
<%!
int a=10,b=20;
String s="welcome to bec"
public int add()
{
return (a+b);
}
public String rever()
{
StringBuffer sb=new StringBuffer(s);
return s.reverse();
}
%>
```

2. Scriptlet Tag:

It includes the java code which we want to embed in service( ) overridden method.

<% code %>

Eg:

```
<%
out.print("Result= "+add());
out.print("Reverse= "+rever());
%>
```

3. Expression Tag:

<%= statement %>

Eg:

<h1>Result= <%= add()%></h1> //To call a method

<%= a%> //To print a variable value

4. Directive Tag:

(@page, @include, @taglib)

<%@page import=”package_name,..” %>

<%@include file=”file_path” %>

<%@taglib prefix=”x” uri=”http://java.sun.com/jsp/jstl/category_name” %>

b) Write a JSP application for conversion of temperature. CO2 L2 7M

Index.html

```
<!DOCTYPE html>
<html>
<head>
        <title>Temperature Converter</title>
</head>
<body>
        <h1>Temperature Converter</h1>
        <form method="post" action="convert.jsp">
                <label for="temperature">Enter Temperature:</label>
                <input type="text" id="temperature" name="t" required>
                <input type="submit" value="Convert">
        </form>
</body>
</html>
```

Convert.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <title>Temperature Conversion Result</title>
</head>
<body>
  <h1>Temperature Conversion Result</h1>
  <%-- JSP Scriptlet --%>
  <%
    String temperatureStr = request.getParameter("t");

    if (temperatureStr != null && !temperatureStr.isEmpty()) {
      double temperature = Double.parseDouble(temperatureStr);
      double convertedTemperature = (temperature * 9/5) + 32;
  %>
  <p>Converted Temperature: <%= convertedTemperature %> Fahrenheit</p>
  <%
    }
  %>
  <a href="index.html">Back to Converter</a>
</body>
</html>
```

(OR)

5 a) List out and explain JSTL tax. CO2 L2 7M

JSTL (JavaServer Pages Standard Tag Library) provides a set of tags that simplify the development of JavaServer Pages (JSP) by offering reusable and standardized functionality. There are several core tags within JSTL, often grouped into the following categories:

1. Core Tags: These tags provide basic control structures and iteration.

- <c:out>: Displays a value after escaping potential HTML characters to prevent cross-site scripting (XSS) attacks.
- <c:set>: Sets a variable's value within the scope.

- <c:remove>: Removes a variable from the scope.
- <c:if>: Conditionally evaluates content based on a specified condition.
- <c:choose>, <c:when>, <c:otherwise>: Simulates switch-case behavior for conditional execution.
- <c:forEach>: Iterates over a collection, such as arrays or collections.

2. Formatting Tags: These tags help format and manipulate data.
- <fmt:formatDate>: Formats a date using specified patterns.
- <fmt:formatNumber>: Formats a number using specified patterns.
- <fmt:bundle>: Resolves messages from a resource bundle.

3. Function Tags: These tags provide utility functions for various tasks.
- <fn:length>: Returns the length of an object.
- <fn:toUpperCase>, <fn:toLowerCase>: Converts strings to uppercase or lowercase.
- <fn:substring>: Retrieves a substring from a string.

4. XML Tags: These tags assist in XML processing.
- <x:parse>: Parses XML documents.
- <x:out>: Outputs XML content.
- <x:set>: Sets XML content.

Example code:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
```

b) Create a Web Application to demonstrate JSF CO2 L3 7M

Example of a JSF web application that displays a simple form to input and display a user's name:

1. Create a new directory for your web application, e.g., "SimpleJSFApp".

2. Inside the "SimpleJSFApp" directory, create a new file named `index.xhtml` (place this file inside the "WebContent" or "WebApp" directory if you're using a standard Maven or Gradle structure):

Jsf code:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
<head>
   <title>Simple JSF Application</title>
</head>
<body>
   <h1>Welcome to Simple JSF Application</h1>
   <h:form>
      <h:outputLabel for="nameInput" value="Enter your name:" />
      <h:inputText id="nameInput" value="#{helloBean.name}" />
      <br/>
      <h:commandButton value="Submit" action="#{helloBean.sayHello}" />
   </h:form>
   <h2>
      <h:outputText value="Hello, #{helloBean.name}!" rendered="#{helloBean.name ne null}" />
   </h2>
</body>
</html>
```

2. Create a Java class named `HelloBean` in a package named `com.example` (adjust the package name according to your organization's convention):

Java Code:

```
package com.example;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
```

```
@ManagedBean
@RequestScoped
public class HelloBean {

   private String name;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }
   public void sayHello() {
      // Method to handle the "Submit" button action
   }
}
```

This example demonstrates a simple JSF application. When you enter your name and click the "Submit" button, it will display a greeting message using the entered name. This is a very basic illustration, but it showcases how JSF manages the user interface and interactions in a Java web application.

Unit-III

6 a) Describe the key features of JAX-RS API that is provided in Java EE? CO3 L2 7M

JAX-RS is a set of Java APIs, provided in Java EE, that are used for the development of RESTful web services.

Following are the key features of the JAX-RS API.

1. POJO-based API - The JAX-RS API provides a set of annotations and corresponding classes and interfaces that can be used with POJOs to expose them as web services.
2. HTTP-based API - The JAX-RS API is designed for HTTP as the base protocol. The API supports HTTP usage patters and provides mapping between HTTP actions and corresponding API classes and annotations.
3. Format Independent - The JAX-RS API is applicable to a wide variety of HTTP entity body content types.
4. Container Independent - JAX-RS applications can be deployed in a Java EE container, Servlet container or can be plugged in as a JAX-WS provider.
5. Dependency Injection: You can use dependency injection to inject resources, such as database connections or EJBs, into your JAX-RS resource classes.
6.Hypermedia Support: While not a core feature, JAX-RS can be extended to support hypermedia controls, enabling the creation of hypermedia-driven RESTful services.
7.Annotations: JAX-RS uses annotations to simplify the creation of RESTful web services. Annotations like @Path, @GET, @POST, @PUT, and @DELETE can be used to define the HTTP methods that correspond to resource URIs.
8.Resource Classes: In JAX-RS, resource classes are Java classes that are annotated with @Path and contain methods annotated with HTTP method annotations (e.g., @GET, @POST). These classes define the resources that will be exposed via RESTful endpoints.
9.URI Path Templates: JAX-RS allows you to define URI path templates using the @Path annotation. You can use placeholders in the path templates to extract values from the request URI and pass them as parameters to your resource methods.
10.Request and Response Objects: JAX-RS abstracts the handling of HTTP requests and responses. You can use javax.ws.rs.core.Request and javax.ws.rs.core.Response objects to work with incoming requests and send responses, respectively.

b) What are request method designator annotations in JAX-RS API? CO3 L2 7M

In JAX-RS (Java API for RESTful Web Services), request method designator annotations are annotations used to specify the HTTP request method that a resource method should handle. These annotations are applied to methods within JAX-RS resource classes and determine which HTTP methods (e.g., GET, POST, PUT, DELETE) should trigger the execution of the annotated method. Here are the main request method designator annotations in JAX-RS:

1.@GET: This annotation is used to indicate that a method should handle HTTP GET requests. It is typically used to retrieve resource representations. The @GET annotation is a

request method designator and corresponds to the similarly named HTTP method.

```
@GET

@Path("/resource")

public Response getResource() {

    // Method logic to handle GET requests

}
```

2.@POST:- The @POST annotation specifies that a method should handle HTTP POST requests. It is commonly used for creating new resources on the server. The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.

```
@POST

@Path("/resource")

public Response createResource(String data) {

    // Method logic to handle POST requests

}
```

3.@PUT:- This annotation designates a method to handle HTTP PUT requests, which are typically used to update existing resources or create a resource if it doesn't exist. The PUT method replaces all current representations of the target resource with the request payload.

```
@PUT

@Path("/resource/{id}")

public Response updateResource(@PathParam("id") int resourceId, String data) {

    // Method logic to handle PUT requests

}
```

4.@DELETE:- The @DELETE annotation marks a method for handling HTTP DELETE requests, which are used to delete resources. The DELETE method deletes the specified resource.

```
@DELETE

@Path("/resource/{id}")

public Response deleteResource(@PathParam("id") int resourceId) {

    // Method logic to handle DELETE requests

}
```

5.@HEAD:- The @HEAD annotation is used to specify that a method should handle HTTP HEAD requests. These requests are similar to GET requests but only retrieve headers and not the actual resource representation.

```
@HEAD
```

```
@Path("/resource/{id}")

public Response getHeadersForResource(@PathParam("id") int resourceId) {

   // Method logic to handle HEAD requests

}
```

6.@OPTIONS:- The @OPTIONS annotation indicates that a method should handle HTTP OPTIONS requests. OPTIONS requests are used to retrieve information about the communication options available for a resource.

```
@OPTIONS

@Path("/resource")

public Response getResourceOptions() {

   // Method logic to handle OPTIONS requests

}
```

(OR)

7 a) Discuss web sockets with an example. CO3 L2 7M

WebSocket is a communication protocol that provides full-duplex, bidirectional communication channels over a single TCP connection. Unlike traditional HTTP, which is request-response-based, WebSocket allows both the client and server to initiate communication independently. This makes WebSocket ideal for real-time applications where low latency and efficient communication are essential, such as chat applications, online gaming, and live data streaming.

Here's an example of using WebSocket in a simple chat application.

Index.html:

```
<!DOCTYPE html>
<html>
<head>
<title>Chat Server</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<script type="text/javascript">
var websocket=new
WebSocket("ws://localhost:8080/ChatRoomServer/chatroomServerEndpoint");
websocket.onmessage=function processMessage(message){
if(message.data!=null)
messageTextArea.value+=message.data+"\n";
}
function sendMessage(){
websocket.send(messageText.value);
messageText.value="";
}
</script>
</head>
<body>
<textarea id="messageTextArea" rows="10" cols="45"
readonly="readOnly"></textarea><br/>
<input type="text" id="messageText" size="50"/>
<input type="button" value="send" onclick="sendMessage()"/>
</body>
</html>
```

ChatRoomServerEndpoint.java:

```
package aaa;
import java.io.IOException;
import javax.websocket.*;
```

```
import javax.websocket.server.ServerEndpoint;
import java.util.*;
@ServerEndpoint("/chatroomServerEndpoint")
public class ChatRoomServerEndpoint {
static Set<Session> chatroomUsers=Collections.synchronizedSet(new
HashSet<Session>());

@OnOpen
public void handleOpen(Session userSession){
chatroomUsers.add(userSession);
}

@OnMessage
public void handleMessage(String message,Session userSession) throws IOException{
String username=(String)userSession.getUserProperties().get("username");
if(username==null){
userSession.getUserProperties().put("username",message);
userSession.getBasicRemote().sendText("You are now connected as:"+message);
}

else{
Iterator <Session> iterator=chatroomUsers.iterator();
while(iterator.hasNext()){
iterator.next().getBasicRemote().sendText(username+" : "+message);
}
}
}

@OnClose
public void handleClose(Session userSession){
chatroomUsers.remove(userSession);
}
}
```

b) Web sockets and Rest API for real time data? Which to choose and why? CO3 L3 7M

Whether you should use a REST API or WebSockets depends on your use case., WebSockets are generally ideal for web apps that require realtime communication, while REST is tailored for retrieving, creating, and managing resources.

When to use WebSockets:

We can broadly group Web Sockets use cases into two distinct categories:

1.Realtime updates, where the communication is unidirectional, and the server is streaming low-latency (and often frequent) updates to the client. Think of live score updates or alerts and notifications, to name just a few use cases.

2.Bidirectional communication, where both the client and the server send and receive messages. Examples include chat, virtual events, and virtual classrooms (the last two usually involve features like live polls, quizzes, and Q&As). WebSockets can also be used to underpin multi-user synchronized collaboration functionality, such as multiple people editing the same document simultaneously.

3.Simplicity: WebSocket is straightforward to implement for real-time scenarios where immediate updates are needed. It simplifies the architecture and reduces the complexity of managing multiple HTTP requests.

Ex:-Telephone communications.

When to use REST:-

REST is ideal for:

- Applications that need to perform actions on resources (e.g. retrieve, update) on an occasional, ad-hoc basis.

- Request-driven, non-realtime, and stateless systems.

- CRUD operations.

- Cloud apps, microservices, and web apps.

1.Statelessness: REST APIs are stateless by design, which means each request from a client to the server must contain all the information needed to understand and process the request. This makes REST APIs suitable for scenarios where stateless interactions are sufficient, such as CRUD operations and retrieving static or cached data.
2.Compatibility: REST is widely adopted and supported by various clients and servers. It's the de facto standard for building web services, and many tools and libraries are available for working with RESTful APIs.
3.Security: REST APIs can be secured using standard HTTP security mechanisms, such as HTTPS and OAuth. Security for WebSocket can be more complex to implement.

Unit-IV

8 a) What is Session bean? What are the types of Session Beans? CO4 L2 7M

A session bean is a type of enterprise Java bean (EJB) that is designed to manage and hold conversational state or data within a specific user session in a Java EE (Enterprise Edition) application. Session beans are used to encapsulate business logic and manage user-specific data during the duration of a session, which can span multiple interactions between a client and the server.

These are further classified into 3 types:

a. Stateless Session beans:
- It can't hold the client's state between calls.
- This bean is indicated with @Stateless annotaion.
- The EJB container will maintain a pool of EJB instances.
- The client requests will be handled by different instances of the EJBs.
- An instance can be shared by multiple clients.
- As soon as the request is served, the client will be released.

b. Stateful Session beans:
- It can hold the client's state between calls.
- This bean is indicated with @Stateful annotaion.
- The EJB container will maintain a single stateful session bean instance for each client.
- That instance will not be shared with the other clients.
- When the communication between the client and the bean ends, the bean will get terminated.
- This client's state will be stored in the instance variables of EJB implementaion class.
- These type of beans are the most commonly used kind of EJB.
- These are used in the applications that collect user data over a sequence of interactions.
- The downside of these beans are, they are not so scalable like stateless beans.

c. Singleton Session beans:
- These are similar to Stateless session beans, but only one instance of Singleton session bean is created.
- That instance doesn't terminates until the application is terminated. -
- This bean is indicated with @Singleton annotaion.
- The created instance is shared between multiple clients and it can be concurently accessed.

b) Explain EJB architecture components. CO4 L2 7M

EJB Architecture:

Java beans incorporate a set of objects into one accessible object that can be accessed easily from any application. This single accessible object is maintainable, customizable, and reusable. The setter/getter method and the single public constructor are used to govern that single accessible object.

Architecture:
The EJB architecture has two main layers, i.e., Application Server and EJB Container, based on which the EJB architecture exist. The graphical representation of the EJB architecture is given below.

Application Server:
In the EJB architecture, the Application server is the outermost layer that holds or contains the Container to be deployed. The application layer plays an important role in executing the application developed using the beans. It provides the necessary environment to execute those applications. Some most popular application servers are Web-logic, Tomcat, JBoss, Web-sphere, Wildfly, and Glass-finish. The main tasks of the application server are:

1. Manage Interfaces
2. Execution of the processes
3. Connecting to the database
4. Manage other resources.

Client Application:
This is the application that interacts with the EJB components. It can be a web application, a desktop application, or any other type of client. The client application accesses EJB components using Remote Method Invocation (RMI) or other protocols.

Remote Method Invocation:
It is a communication protocol that allows the client application to invoke methods on EJB components running on a remote server. The client application communicates with the EJB container using RMI or other protocols.

EJB Container: The EJB container provides a runtime environment for executing EJB components. It manages the lifecycle of EJB instances, handles remote method invocations, and provides various services like transaction management, security, concurrency, and pooling of EJB instances.

BEANS
Java beans of the enterprise are installed in the Container in the same way as a Plain old java object (POJO) is installed and registered to the Container. For developing secured, large scale and robust business applications, beans provide business logic.

Types Of Ejbs:
There are three types of Enterprise Java Beans or EJB available, which are as follows:

- Stateless Enterprise Java Beans
- Stateful Enterprise Java Beans
- Message-driven Enterprise Java Beans

Stateless EJB
In order to implement the stateless business logic, the stateless EJBs are primarily used. Storing a user's physical address into an inventory system's database is an example of the stateless EJB.

Stateful EJB
The Stateful EJB is just opposite to the Stateless EJB. The Stateful EJBs are used when we have to maintain the state of the application on the backend during the user session. The shopping cart of an online shopping application is an example of the Stateful EJB.

Message-driven EJB
Another special type of EJB that is used for sending and receiving messages from message brokers implements the JMS specification. The systems based on the broker are loosely coupled. The components that communicate through the broker have the advantage of not waiting for one request to finish before submitting another request because the broker by nature is asynchronous.

(OR)

9 a) Explain the advantages of Multithreading over Enterprise Beans CO4 L2 7M

Multi-threading and Enterprise Beans are two different concepts used in application development, each with its own advantages and use cases. Let's compare their advantages:

Advantages of Multi-threading:

1. **Parallel Execution**: Multi-threading allows different threads to execute concurrently, making the most of available CPU cores and potentially improving performance for tasks that can be parallelized.
2. **Responsive User Interfaces**: In client applications, multi-threading can keep the user

interface responsive by offloading time-consuming tasks to background threads, preventing the UI from freezing.
3. **Resource Utilization**: Multi-threading enables efficient utilization of resources in applications that need to perform multiple tasks simultaneously.
4. **Customized Concurrency**: Developers have direct control over thread creation, scheduling, and synchronization, allowing them to fine-tune concurrency for specific scenarios.
Advantages of Enterprise Beans:
1. **Managed Concurrency**: Enterprise Beans (EJBs) are managed by the EJB container, which handles threading and concurrency issues. This simplifies development and reduces the likelihood of thread-related bugs.
2. **Scalability and Distribution**: EJBs are designed for distributed and scalable applications. They can be deployed across different servers and managed by the container, which handles load balancing and failover.
3. **Transaction Management**: EJB containers provide built-in support for transaction management, ensuring data integrity and consistency in complex applications.
4. **Security and Resource Management**: EJB containers handle security aspects, resource pooling, and connection management, allowing developers to focus on business logic rather than low-level concerns.
In summary, multi-threading and Enterprise Beans have their own advantages based on the requirements of the application. Multi-threading is useful for applications that require low-level control over concurrency, parallelism, and responsiveness, especially in client-side scenarios. Enterprise Beans are suitable for building scalable, distributed, and transactional applications, with built-in support for concurrency management, security, and resource management.

b) Explain different application of session beans. CO4 L2 7M

Session beans are one of the three types of Enterprise JavaBeans (EJBs) and are commonly used in Java EE applications to encapsulate business logic and provide various services. Here are the different applications of session beans:
1. **Business Logic Components**: Session beans are often used to encapsulate the core business logic of an application. They can process data, perform calculations, execute complex algorithms, and manage interactions with databases and other external systems.
2. **Service Facades**: Session beans can act as service facades that provide simplified interfaces for clients to access various application services. They aggregate multiple lower-level services into a higher-level, user-friendly API.
3. **Security and Access Control**: Session beans can enforce security and access control rules. They can authenticate users, authorize access to certain functionality, and enforce permissions.
4. **Transaction Management**: Session beans can manage transactions, ensuring that multiple operations are grouped into a single transaction. They provide support for atomicity, consistency, isolation, and durability (ACID properties) of transactions.
5. **Caching and Optimization**: Stateful session beans can be used to maintain client-specific state and cache data, reducing the need to repeatedly fetch data from databases. This can improve performance by reducing database access overhead.
6. **Integration with External Systems**: Session beans can act as integration points with external systems. They can encapsulate communication protocols and data transformation required to interact with external services.
7. **Workflow and Process Management**: Session beans can model and manage workflows and processes in an application. They can guide users through multi-step processes and orchestrate different components of the application.
8. **Message-Driven Components**: Message-driven beans, a type of stateless session bean, can be used to asynchronously process messages in applications that require decoupled communication, such as event-driven architectures or messaging systems.
9. **Data Transformation and Validation**: Session beans can validate and transform incoming data from clients, ensuring data consistency and integrity before processing.
10. **Web Services**: Session beans can be exposed as web services, allowing remote clients to invoke methods via SOAP or RESTful endpoints, enabling integration with other applications.
In summary, session beans are versatile components that offer a wide range of applications,

from encapsulating business logic to managing transactions, security, caching, and integration with external systems. They play a pivotal role in building robust, scalable, and modular Java EE applications.

Prepared by Head of the Department