18IT503

Hall Ticket Number:

III/IV B.Tech (Regular\Supplementary) DEGREE EXAMINATION

Feb, 2021 — **Information Technology**

Fifth Semester — **Enterprise Programming**

Time: Three Hours — **Maximum:** 50 Marks

Answer all questions from Part-A. (1X10= 10 Marks)

Answer ANY FOUR questions form Part-B. (4X10=40 Marks)

Part-A

1. Answer all questions (1X10=10 Marks)

a)	What are the various Java EE Applications?	CO1
b)	What is difference between Get and Post method?	CO1
c)	What is servlet?	CO1
d)	List the JSP standard tags.	CO2
e)	Write the purpose of web socket encoder.	CO2
f)	List the JSP elements?	CO2
g)	List the services provided by EJB.	CO3
h)	Differentiate between session bean and entity bean.	CO3
i)	What are the disadvantages using EJB?	CO3
j)	What is SOAP?	CO4

Part-B

UNIT-I

2.	a)	Explain about java EE architecture.	[CO1] 5M
	b)	Write the differences between ServletConfig and ServletContext classes.	[CO1] 5M
3.	a)	Write a shot note on servlet life cycle with an example.	[CO1] 5M
	b)	Write a servlet program to demonstrate the http parameters.	[CO1] 5M

UNIT-II

4.	a)	Draw and describe the JSP life cycle.	[CO2] 5M
	b)	Write a JSP program to develop the student registration page.	[CO2] 5M
5.	a)	Explain the architecture of a JSF Application with an example?	[CO2] 5M
	b)	Write a short note on Web socket life cycle.	[CO2] 5M

UNIT-III

6.	a)	List and explain the types of EJB.	[CO3] 5M
	b)	Draw and explain the life cycle of EJB.	[CO3] 5M
7.	a)	Write about java persistent query language.	[CO3] 5M
	b)	Write an application to demonstrate the EJB.	[CO3] 5M

UNIT-IV

8.	a)	Explain SOAP web services.	[CO4] 5M
	b)	Explain about any one web server.	[CO4] 5M
9.	a)	Explain WSDL. Write different WSDL elements and attributes.	[CO4] 5M
	b)	Write an application to create SOAP service?	[CO4] 5M

1. Answer all questions (1X10=10 Marks)

a) What are the various Java EE Applications? CO1

The various Java EE applications are web components and EJB components. Servlets, JSP's, JSF's are web components. Session beans and Message Driven beans are EJB components.

b) What is difference between Get and Post method? CO1

In get method the data is exposed in URL bar but whereas in Post method, the data is not exposed in URL bar

c) What is servlet? CO1

A Servlet is a Java object that processes the server side of HTTP interactions. The Web container or servlet container is responsible to create the object of servlet

d) List the JSP standard tags. CO2

<c:out>, <c:set>, <c:remove> ,<c:catch>, <c:if>, <c:choose>, <c:when>

[Any Four]

e) Write the purpose of web socket encoder. CO2

An web socket encoder takes a java object and produces a representation that can be transmitted as a web socket message.

f) List the JSP elements? CO2

1.Scriplets
2.Declarations
3.Expression
4.Commentts
5.Directives

g) List the services provided by EJB. CO3

1)Lifecycle
2)Security
3)Concurrency
4)Transaction
5)Asynchrony

h) Differentiate between session bean and entity bean. CO3

Session Beans represent a process or flow whereas Entity bean represent row in a database. Each session bean is associated with one EJB client at time. Each entity bean is associated with more than one client at time.

[Any two valid points]

i) What are the disadvantages using EJB? CO3

1) Requires application server.
2) Requires only java client, for other language client you need to go for web service.
3) Complex to understand and develop EJB application.

j) What is SOAP? CO4

SOAP stands for Simple Object Access Protocol. It is a XML-based protocol for accessing web services.
It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications.

2. a) Explain about java EE architecture. [CO1] 5M
[Diagram-2M, Explanation-3]

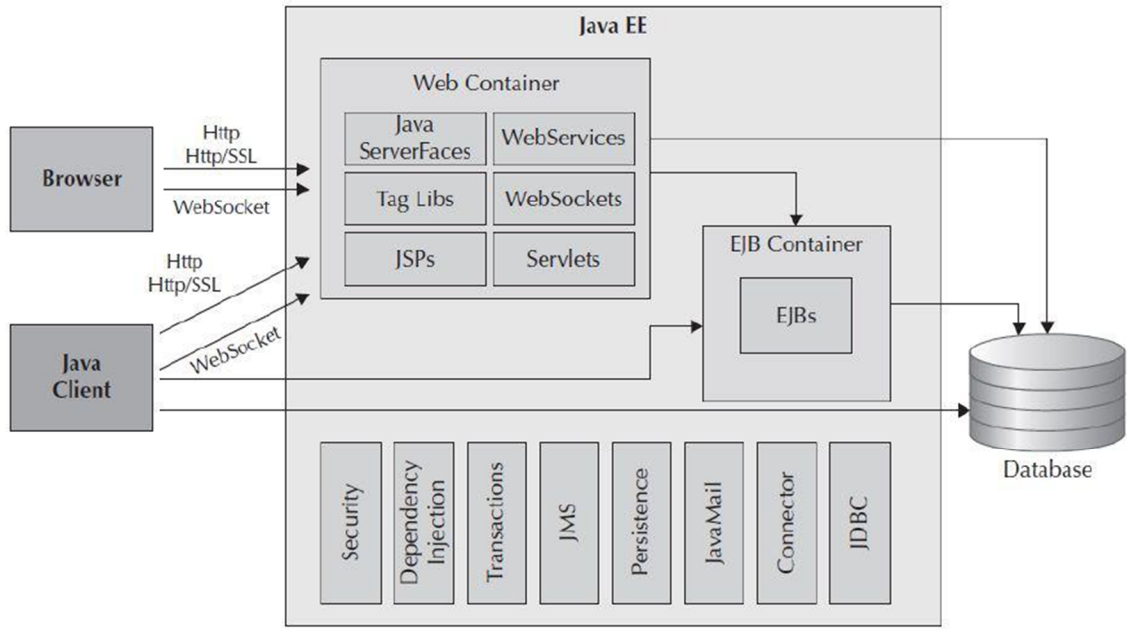


FIGURE 1-1. *Architecture of the Java EE platform*

The Java EE platform supports a wide variety of protocols that clients may use to interact.

- **Browser client:** This connects to the Java EE application using standard web protocols such as HTTP and WebSockets.
- **Java clients:** This connects by Java EE application such as Java desktop application, running on a different application server.

Java EE platform refers to the runtime environment provided by a Java EE application server.

- **Java EE container**, the container can mediate or intercept calls to and from the application code, and insert other kinds of logic that qualify and modify the calls to and from the application code.
- The Java EE container can enforce security rules on the application that is running, for example, a rule such as "only allow access from suresh and ramesh to my application."
- The Java EE server container is itself made up of two other containers:
    - Web container
    - Enterprise JavaBeans (EJB) container.
- The **web container** is devoted to running the web components in a Java EE application: the web pages, Java servlets, and other Java EE web components that can interact with clients connecting to the Java EE application with standard web protocols.
- The **EJB container** is devoted to running the application logic part of the Java EE application. Enterprise JavaBeans are Java classes that contain and manipulate the core data structures of the Java EE application.

These other boxes represent a variety of services that a Java EE application may choose to use.

- The **security** service enables to restrict access to its functions to only a certain set of known users.
- The **dependency injection** service enables to delegate the lifecycle management and discovery of some of its core components.
- The **transaction** service enables to define collections of methods that modify application data in such a way that either all the methods must complete successfully, or the whole set of method executions is rolled back as though nothing has ever happened.
- The **Java Message Service (JMS)** exposes to the ability to reliably send messages to other servers in the deployment environment of the Java EE application server.

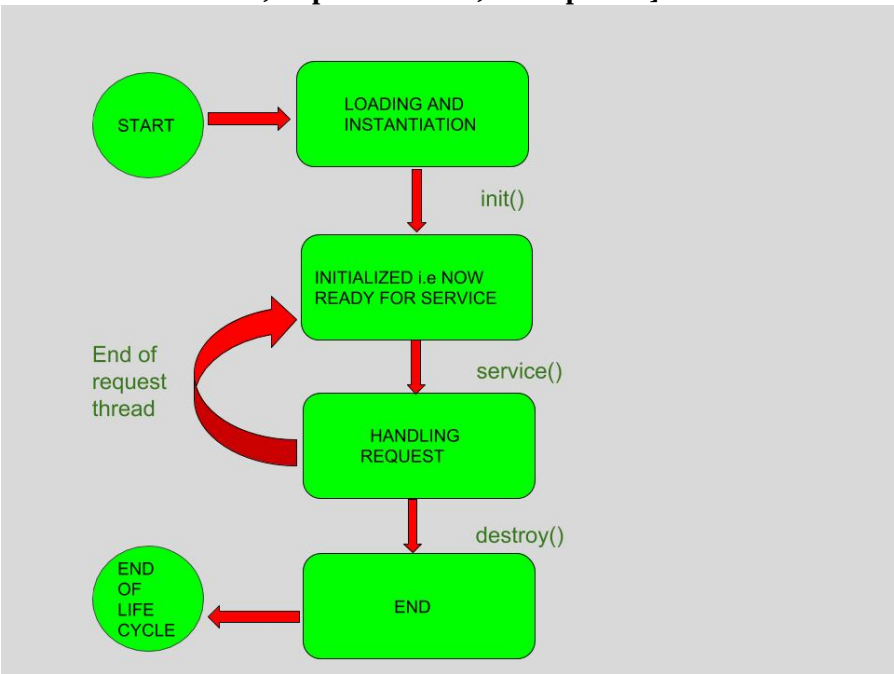
- The **Persistence** service enables application data in the form of a Java object to be synchronized with its equivalent form in the tables of a relational database.
- The **JavaMail**service enables to send email, particularly useful in the kind of application that takes some action initiated by and on behalf of a user, and which needs to notify the user at some later time of the outcome of the action.
- The **Java EE Connector Architecture (JCA)**, which provides a framework into which a new service that is not a standard part of the Java EE platform may be added and that can then, in turn, be utilized by a Java application running in the platform.
- The **Java Database Connectivity (JDBC)** API supports traditional storage and retrieval of Java EE application data in a relational database using the SQL query language.

The database tier of the Java EE platform holds all application data that the Java EE application needs to exist longer than the scope of a single session of the application.

2 b) Write the differences between ServletConfig and ServletContext classes. [CO1] 5M

ServletConfig	ServletContext
ServletConfig is servlet specific	ServletContext is for whole application
Object of ServletConfig will be created during initialization process of the servlet	Object of ServletContext will be created at the time of web application deployment
ServletConfig object is obtained by getServletConfig() method.	ServletContext object is obtained by getServletContext() method.
Each servlet has got its own ServletConfig object.	ServletContext object is only one and used by different servlets of the application.
Parameters of servletConfig are present as name-value pair in <init-param> inside <servlet>.	Parameters of servletContext are present as name-value pair in <context-param> which is outside of <servlet> and inside <web-app>
Use ServletConfig when only one servlet needs information shared by it.	Use ServletContext when whole application needs information shared by it
We should give request explicitly, in order to create ServletConfig object for the first time	ServletContext object will be available even before giving the first request
Syntax and Example	**Syntax and Example**

3. a) Write a shot note on servlet life cycle with an example **[CO1] 5M**
[Diagram and method list - 1M, Explanation-2.5, Example-1.5]



The methods in servlet life cycle are:

- **init()** method.
- **service()** method.
- **destroy()** method.

init() Method

- The servlet is initialized by calling the **init()** method.
- The init method is called only once.
- It is called only when the servlet is created, and not called for any user requests afterwards.
- Syntax:

```
public void init() throws ServletException     {
        // Initialization code...
}
```

service() Method

- The servlet calls **service()** method to process a client's request.
- It is the main method to perform the actual task.
- service() method to handle requests coming from the client (browsers) and to write the formatted response back to the client.
- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- Syntax:

```
public void service (ServletRequest request,ServletResponse response)
   throws ServletException, IOException
   {
            //body
   }
```

destroy() Method

- The servlet is terminated by calling the **destroy()** method.
- The destroy() method is called only once at the end of the life cycle of a servlet.
- Syntax:

```
public void destroy()
{
        // Finalization code...
 }
```

[Any valid example]

3 b) Write a servlet program to demonstrate the http parameters. [CO1] 5M
[HTML Page – 1M, Web.xml file – 1M, Servlet Page – 2M, Output – 1M]

HTML Page

```
<html>
  <body>
    <form action = "HelloForm" method = "GET">
      First Name: <input type = "text" name = "first_name">
      <br />
      Last Name: <input type = "text" name = "last_name" />
      <input type = "submit" value = "Submit" />
    </form>
  </body>
</html>
```

[Any valid HTML Page]

Web.xml

```
<servlet>
  <servlet-name>HelloForm</servlet-name>
  <servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>HelloForm</servlet-name>
  <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

Servlet Page

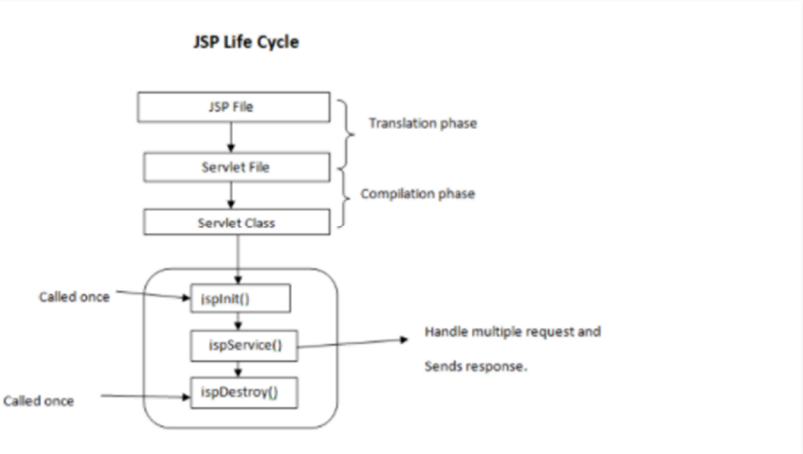
```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
    String title = "Using GET Method to Read Form Data";
    String docType =
      "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";

    out.println(docType +
      "<html>\n" +
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor = \"#f0f0f0\">\n" +
          "<h1 align = \"center\">" + title + "</h1>\n" +
```

```
            "<ul>\n" +
              "  <li><b>First Name</b>: "
              + request.getParameter("first_name") + "\n" +
              "  <li><b>Last Name</b>: "
              + request.getParameter("last_name") + "\n" +
            "</ul>\n" +
          "</body>" +
        "</html>"
      );
  }
}
```

[Any Valid Example Program]

4. a) Draw and describe the JSP life cycle. [CO2] 5M
[Diagram and steps - 1M, Explanation-2.5, Example-1.5]

- This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.
- Steps 1. Compilation, 2. Initialization, 3. Execution, 4. Cleanup



1. Compilation:

- If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.
- The compilation process involves three steps −
  a. Parsing the JSP.
  b. Turning the JSP into a servlet.
  c. Compiling the servlet.

2. Initialization:

- When a container loads a JSP it invokes the **jspInit()** method before servicing any requests.
- Initialization is performed only once and as with the servlet init method.
- generally initialize database connections, open files, and create lookup tables in the jspInit method.
- Syntax:

```
public void jspInit() {
  // Initialization code...
}
```

3. Execution:

- Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.
- The **_jspService()** method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows −
- The **_jspService()** method of a JSP is invoked on request basis.
- This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, **GET, POST, DELETE**, etc.
- Syntax:

```
void _jspService(HttpServletRequest request, HttpServletResponse response)
 {
        // Service handling code...
 }
```

4. Cleanup:

- The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets.
- Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.
- Syntax:

```
public void jspDestroy() {
  // Your cleanup code goes here.
}
```

4 b) Write a JSP program to develop the student registration page. [CO2] 5M
[Login page – 1.5M, Registration Page- 2.5M, Output – 1M]

login.jsp

```
<%@ include file="index.jsp" %>
<hr/>
<h3>Login Form</h3>
<%
String profile_msg=(String)request.getAttribute("profile_msg");
if(profile_msg!=null){
out.print(profile_msg);
}
String login_msg=(String)request.getAttribute("login_msg");
if(login_msg!=null){
out.print(login_msg);
}
 %>
 <br/>
<form action="loginprocess.jsp" method="post">
Email:<input type="text" name="email"/><br/><br/>
Password:<input type="password" name="password"/><br/><br/>
<input type="submit" value="login"/>"
</form>
```

Registration page

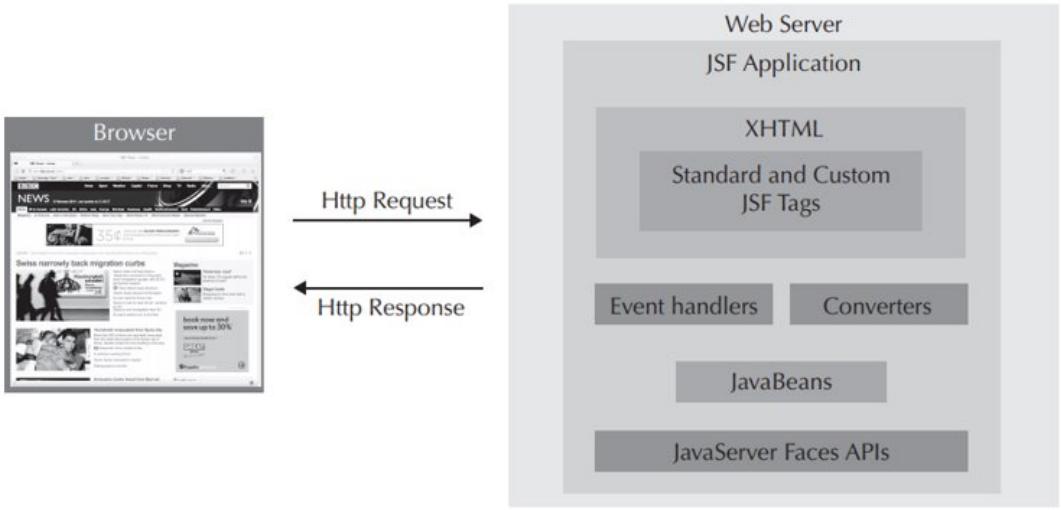
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Guru Registration Form</title>
</head>
<body>
<h1>Guru Register Form</h1>
<form action="guru_register" method="post">
<table style="with: 50%">
<tr>
<td>First Name</td>
<td><input type="text" name="first_name" /></td>
</tr>
<tr>
<td>Last Name</td>
<td><input type="text" name="last_name" /></td>
</tr>
<tr>
<td>UserName</td>
<td><input type="text" name="username" /></td>
</tr>
<tr>
<td>Password</td>
<td><input type="password" name="password" /></td>
```

```
</tr>
<tr>
<td>Address</td>
<td><input type="text" name="address" /></td>
</tr>
<tr>
<td>Contact No</td>
<td><input type="text" name="contact" /></td>
</tr></table>
<input type="submit" value="Submit" /></form>
</body>
</html>
```

[Any valid programs with output]

5. a) Explain the architecture of a JSF Application with an example? [CO2] 5M
[Diagram – 2M, Explanation – 3M]

- A JSF application is made up of a collection of
  - XHTML pages
  - Java classes
  - JSF metadata.



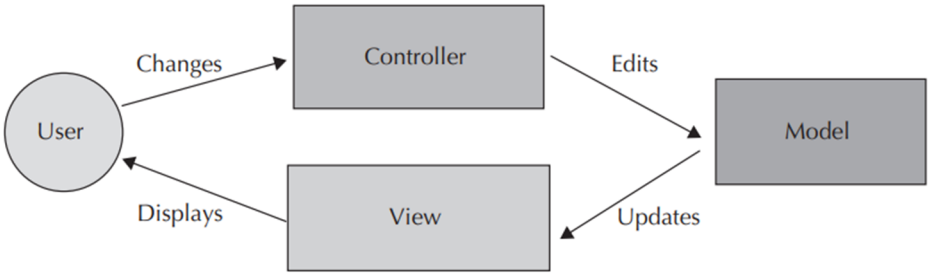
- The **XHTML pages** make up the visual part of the web application.
- It holding the various **JSF tags, markup, and scriptingcontent** that define the UI elements.
- The **Java classes** in the JSF application define the **application data** that is housed in the web application, and also **mediate the way the application data** is bound into the presentation layer of the application.
- The **Java classes** include JavaBeans that hold various parts of the application data.
- Together with Java classes that mediate the data by implementing various types from the JSF APIs, including event handlers, data validators, and data converters.
- The **JSF metadata** defines how the XHTML pages and JavaBeans are treated in the JSF runtime.
- This metadata defines the scope of the JavaBeans, how many instances are used, and their lifecycles, together with navigation rules, and can be used to control how the UI components are rendered.

Model-View-Controller

- The **user** makes a change to the UI, which is passed to the controller.
- The **controller**'s job in this design pattern is to interpret the changes to the data model that the user

is asking for and transform that request, or series of requests, into a form that can be used to edit the data model.

- When the model changes, the controller's job is to make sure that the view is informed of the change.
- The **view**, in turn, has the task of altering its state in such a way that the display information it presents to the user of the application is consistent with the newly edited state of the data model.



- Using this kind of separation of concerns, the data model never needs to interpret all the different ways that a user may ask for a change to the data model, because the controller is taking care of that.
- The data model need never concern itself with how the data model is displayed to the user, because the view is taking care of that.

Example:

Index.xhtml:

```
<?xml version='1.0' encoding='UTF-8' ?>
 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
 <html xmlns=http://www.w3.org/1999/xhtml
       xmlns:h="http://xmlns.jcp.org/jsf/html">
  <head>
    <title>Simplest JSF Page</title>
  <head>
  <body>
    <div align="center">
      <br />
      <h:form>
      <h:inputText value="#{myHelloBean.name}"/>
      </h:form>
      <br />
      Hello to you, #{myHelloBean.name} !
    </div>
  </body>
 </html>
```

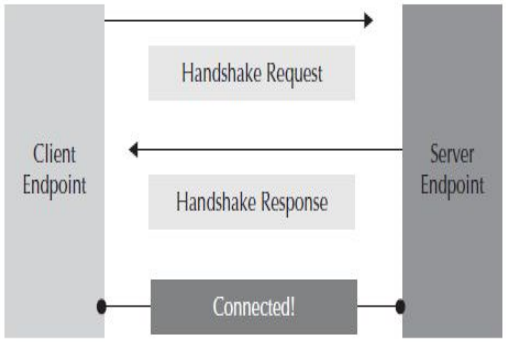
HelloBean.java:

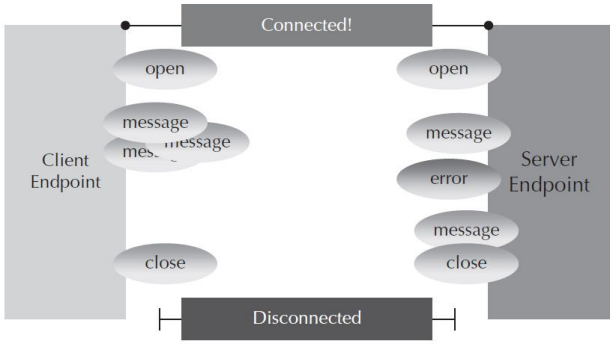
```
import javax.inject.Named;
import javax.enterprise.context.*;

@Named(value = "myHelloBean")
@RequestScoped
public class HelloBean {
  private String name = "dear reader";

  public void setName(String name) {
     this.name = name;
  }

  public String getName() {
     return this.name;
  }
}
```

5 b) Write a short note on Web socket life cycle. [CO2] 5M
[Diagram – 2M, Explanation – 3M]

Web Socket life cycle:

- First, the client initiates a connection request.
- This occurs when the client sends a specially formulated HTTP request to the web server.
- What identifies this as a WebSocket opening handshake request over any common or garden-variety HTTP request is the use of the ***Connection : Upgrade and Upgrade : websocket***headers, and the most important information is the request URI, /myChat.
- The web server decides whether it supports WebSockets at all (all Java EE web containers do).
- If so, whether there is an endpoint at the request URI of the handshake request that meets the requirements of the request.
- If all is well, the WebSocket-enabled web server responds with an equally specially formulated HTTP response called a WebSocket opening handshake response
- This response confirms that the server will accept the incoming TCP connection request from the client and may impose restrictions on how the connection may be used.
- Once the client has processed the response and is happy to accept any such restrictions, the TCP connection is created.
- Each end of the connection may proceed to send messages to the other.





Once the connection is established, a number of things can occur:

- ***Either end of the connection may send a message to the other.*** This may occur at any time that the connection is open. Messages in the WebSocket protocol have two flavors: **text and binary**.
- ***An error may be generated on the connection.*** In this case, assuming the error did not cause the connection to break, both ends of the connection are informed. Such non terminal errors may occur, for example, if one party in the conversation sends a badly formed message.
- ***The connection is voluntarily closed.*** This means that either end of the connection decides that the conversation is over and so closes the connection. Before the connection is closed, the other end of the connection is informed of this.

Life cycle - @onOpen, @onClose, @onError, @onMessage, with explanation.

6. a) List and explain the types of EJB. [CO3] 5M
[List – 1M, Explanation – 4M]

There are **three** types of EJB:

1. Session beans
2. Entity Bean
3. Message-driven beans

1. Session beans

- Session bean encapsulates business logic only, it can be invoked by local, remote and webservice client.
- It can be used for calculations, database access etc.
- The life cycle of session bean is maintained by the application server (EJB Container).
- These are not persistent because they do not survive a server crash or a network failure.
- There are three types of Session beans:
  - **A) Stateful Session Bean**
  - **B) Stateless Session Bean**
  - **C) Singleton Session Bean**

A) Stateful Session Bean:

- It maintains state of a client across multiple requests.
- Stateful session bean can be used to access various method calls by storing the information in an instance variable.
- in which it is necessary to maintain state, such as instance variable values or transactional state, between method invocations.
- For example, shopping site, the items chosen by a customer must be stored as data is an example of stateful session bean.
- A stateless session bean implementation class is marked **@Statefull.**

B) Stateless Session Bean

- It doesn't maintain state of a client between multiple method calls.
- Stateless session bean can be used in situations where information is not required to used across call methods.
- Stateless session beans do not share state or identity between method invocations.
- They are useful mainly in middle-tier application servers that provide a pool of beans to process frequent and brief requests.
- A stateless session bean implementation class is marked **@Stateless.**

C) Singleton Session Bean

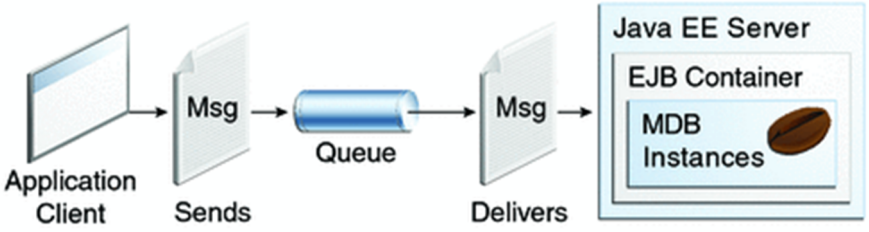
- One instance per application, it is shared between clients and supports concurrent access.
- A stateless session bean implementation class is marked **@Singleton.**

2. Entity Bean

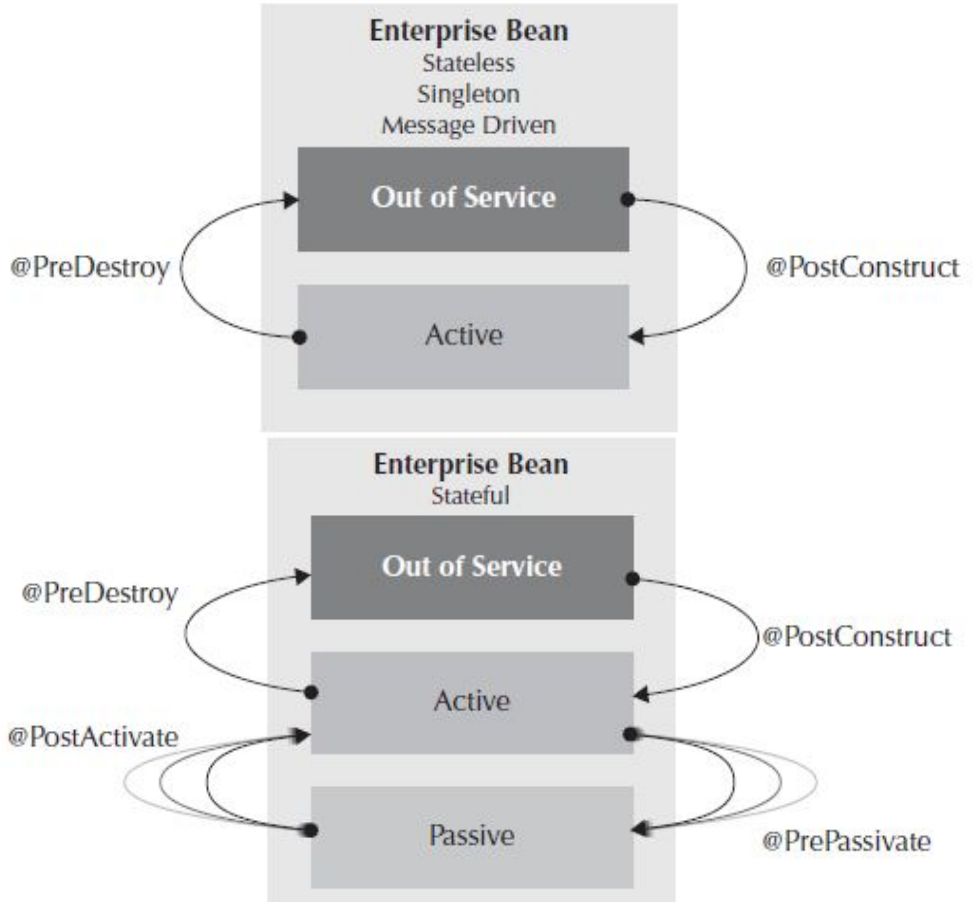
- Entity bean represents the persistent data stored in the database. It is a server-side component.
- There was two types of entity beans: **bean managed persistence** (BMP) and container managed persistence (CMP).
- it is deprecated and replaced by JPA (Java Persistence API) that is covered in the hibernate tutorial.

3. Message-driven beans

- Message-Driven Beans (MDB) provide an easier method to implement asychronous communication than using straight JMS.
- MDBs were created to receive asynchronous JMS messages.
- The container handles much of the setup required for JMS queues and topics. It sends all messages to the interested MDB.
- They are marked using the @MessageDriven annotation



6 **b) Draw and explain the life cycle of EJB.** **[CO3] 5M**



- Singleton and stateless session beans together with message-driven beans have the same lifecycle.
- There are two states: Out of Service and Active.
- When beans are being brought into service, they start in the Out of Service state.
- When the container needs to bring a bean into service, it invokes the constructor, injects any dependencies (@EJB annotation) and then brings the instance into the Active state.
- The bean instance can accept and service incoming calls from clients.
- When the container has no more need for the bean instance, it brings the instance out of service.
- cleans up any supporting resources, and releases the bean instance for garbage collection.
- Stateless session beans and message-driven bean instances are required to exist only for the
- duration of a client call.
- Enterprise Bean containers may bring them into service only as a client makes a call to them.
- Enterprise Bean containers may destroy such bean instances as soon as the client call is serviced.
- singleton session beans are created only once prior to any client calls being serviced, and then destroyed only once all the client calls have been completed, when the container shuts down.
- You may intercept all these state changes on your Enterprise Beans, from Out of Service to Active, and from Active to Out of Service.
- Java method to be called when the state change occurs and mark it with one of the state change annotations.
- when it is moving from the Out of Service to the Active state, you use the @PostConstruct annotation.
- Note: the bean instance has been constructed and any dependency injection has already occurred.
- when it is moved from the Active state to the Out of Service state, you use the @PreDestroy annotation.
- Stateful session beans have a lifecycle that includes these same states, Out of Service, Active and Passive state.
- use @PostConstruct and @PreDestroy annotations to ask the container to call you when it transitions your Enterprise Beans between these states.
- The Passive state is one in which the Enterprise Bean instance is taken out of service temporarily.
- The container will not pass any client calls to the Enterprise Bean while it is in this state.

- When a call comes in for a bean in the Passive state, the container brings it back into the Active state in order to respond to the call.
- The container may bring the same Stateful session bean instance in and out of the Passive state many times.
- The reason for this extra state in the case of Stateful session beans is, the container must create a new instance for each and every client that uses it.
- I.e., stateful session beans with large numbers of clients, the container needs to support correspondingly large numbers of instances.
- using the @PrePassivate and @PostActivate events to annotate methods so the container calls them during the transitions.
- Final note on singleton beans: the Enterprise Bean container has a choice as to when to instantiate.
- It can do it at any time between the deployment of the application containing the singleton bean and the moment immediately prior to a client call needing a response.
- The class-level @Startup annotation on a singleton bean to direct the container to instantiate the bean upon application deployment, rather than allow the container to leave it to the last minute.

7. a) Write about java persistent query language. [CO3] 5M

[Description – 3, Example – 2M]

- **The Java Persistence API defines an SQL.**
- **The basic statements are SELECT , UPDATE , and DELETE.**
- **JPA queries are created with the EntityManager in two primary ways.**
- **1. dynamically using the createQuery() API**
- **Query q = myEntityManager.createQuery("SELECT a.firstName, a.lastName FROM Author a ORDER BY a.lastName");**

. named queries

- **Named queries are precompiled JPQL statements that are declared using the @NamedQuery annotation.**
- **For example, the following Author entity holds a named query with the name findAuthors :**

```
@Entity
    @NamedQuery(
        name="findAuthors",
        query="select a from Author a"
    )
public class Author implements Serializable {
    @Id
    private int id;
    ...
}
```

- **When the JPA application wishes to call the query, it creates the Query object using the createNamedQuery() method on the EntityManager , passing in the name of the named query.**
- **For example,**

Query q = myEntityManager.createNamedQuery("findAuthors");

- **CREATE or an UPDATE statement query is executed by calling executeUpdate() method**
- **If the query is a SELECT statement query is executed by calling either**
- **List getResultList() method, which returns a List of result objects of the type expected when multiple are expected,**
- **Object getSingleResult() method, which returns a single result object of the type expected in the case when a single result is expected.**

[Any valid example program]

7 b) Write an application to demonstrate the EJB. [CO3] 5M
[Program – 4M,Output-1M]

The HelloBean remote interface

```
import javax.ejb.Remote;
@Remote
public interface HelloBean {
public String getMessageFor(String caller);
}
```

The HelloBean implementation class

```
import javax.ejb.Stateful;
@Stateful
public class HelloBeanImpl implements HelloBean {
@Override
public String getMessageFor(String caller) {
return "hello to you, " + caller + " !";
}
}
```

The HelloBean client: ServletClient

```
import java.io.IOException;
import java.io.PrintWriter;
import javaeems.chapter9.hello.HelloBean;
import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet(name = "ServletClient", urlPatterns = {"/ServletClient"})
public class ServletClient extends HttpServlet {
@EJB
private HelloBean helloBean;
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
try {
out.println("<html>");
out.println("<head>");
out.println("<title>Hello Java EE</title>");
out.println("</head>");
out.println("<body>");
out.println("<br>");
out.println("<div align='center'>");
out.println("<h2>Hello Enterprise Beans</h2>");
out.println("<br>");
String displayMessage;
String message = helloBean.getMessageFor("web client");
out.println("Result of calling the enterprise bean: <br><b>"
```

```
+ message + "</b>");
out.println("</div>");
out.println("</body>");
out.println("</html>");
} finally {
out.close();
}
}
}
```

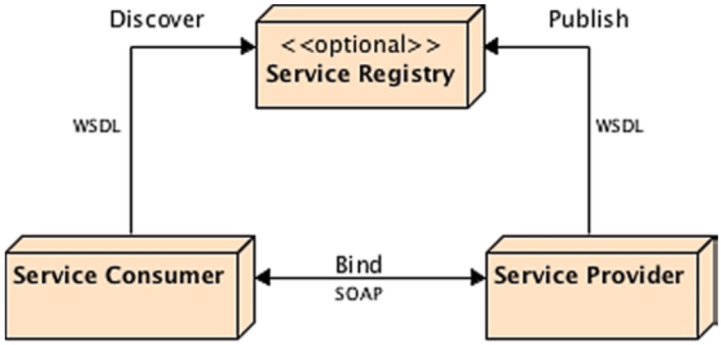
Output:

8. a) Explain SOAP web services. **[CO4] 5M**

[Diagram – 1.5, Description – 3.5]

SOAP is an XML-based protocol for accessing web services over HTTP. It has some specification which could be used across all applications.

The SOAP web service can optionally register its interface into a registry (Universal Description Discovery and Integration, or UDDI) so a consumer can discover it. Once the consumer knows the interface of the service and the message format, it can send a request to the service provider and receive a response.



SOAP web services depend on several technologies and protocols to transport and to transform data from a consumer to a service provider in a standard way. The ones that you will come across more often are the following:

Extensible Markup Lan • guage (XML) is the basic foundation on which SOAP web services are built and defined (SOAP, WSDL, and UDDI).

• Web Services Description Language (WSDL) defines the protocol, interface, message types, and interactions between the consumer and the provider.

• Simple Object Access Protocol (SOAP) is a message-encoding protocol based on XML technologies, defining an envelope for web services communication.

• Messages are exchanged using a transport protocol. Although Hypertext Transfer Protocol (HTTP) is the most widely adopted transport protocol, others such as SMTP or JMS can also be used.

• Universal Description Discovery, and Integration (UDDI) is an optional service registry and discovery mechanism, similar to the Yellow Pages; it can be used for storing and categorizing SOAP web services interfaces (WSDL).

With these standard technologies, SOAP web services provide almost unlimited potential. Clients can call a service, which can be mapped to any program and accommodate any data type and structure to exchange messages through XML.

8 b) Explain about any one web service. **[CO4] 5M**

Web services:

1. **JAX-WS**: for SOAP web services
2. **JAX-RS**: for RESTful web services

Explanation of anyone web service with an example.

9. a) Explain WSDL. Write different WSDL elements and attributes. [CO4] 5M

[Definition - 1, Elements – 2.5, Attributes – 2.5]

WSDL stands for Web Service Description Language. WSDL is an XML based document that provides technical details about the web service. Some of the useful information in WSDL document are: method name, port types, service end point, binding, method parameters etc

Some of the different tags in WSDL xml are:

xsd:import namespace and schemaLocation: provides WSDL URL and unique namespace for web service.
message: for method arguments
part: for method argument name and type
portType: service name, there can be multiple services in a wsdl document.
operation: contains method name
soap:address for endpoint URL.

Explanation in detail

9 b) Write an application to create SOAP service? [CO4] 5M

[HTML Page – 1M, Servlet Page – 4M]

INDEX.HTML

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Calculator Service</title>
   </head>
   <body>
      <h1>Calculator Service</h1>
      <form name="Submit" action="ClientServlet"> 
         <input type="text" name="value1" value="2" size="3"/>+
         <input type="text" name="value2" value="2" size="3"/>=
         <input type="submit" value="Get Result" name="GetResult" />
      </form>
   </body>
</html>
```

ClientServlet.java

```
package org.me.calculator.client;
import java.io.*;
import javax.annotation.Resource;
        import javax.servlet.*;
        import
        javax.servlet.annotation.WebServlet;
        import javax.servlet.http.*;
        import
        javax.xml.ws.WebServiceContext;
        import
        javax.xml.ws.WebServiceRef;
        /**
         *
         * @author mg116726
```

```
 */
@WebServlet(name="ClientServlet",
urlPatterns={"/ClientServlet"})public class ClientServlet extends
HttpServlet {
    @WebServiceRef(wsdlLocation                                                    =
"http://localhost:8080/CalculatorApp/CalculatorWSService?wsdl")
    public CalculatorWSService service;
    @Resource
    protected WebServiceContext context;
    /**
    * Processes requests for both HTTP <code>GET</code> and <code>POST</code> methods.
    * @param request servlet request
    * @param response servlet response
    */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-
        8");PrintWriter out = response.getWriter();
        try {
            out.println("<h2>Servlet ClientServlet at " + request.getContextPath () + "</h2>");

                org.me.calculator.client.CalculatorWS port = service.getCalculatorWSPort();

                int i =
                Integer.parseInt(request.getParameter("value1"));
                int j =
                Integer.parseInt(request.getParameter("value2"));
                int result = port.add(i, j);
                out.println("<br/>");
                out.println("Result:");
                out.println("" + i + " + " + j + " = " + result);
                ((Closeable)port).close();
        }
            finall
            y {
            out.c
            lose(
            );
        }
    }
    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign
on theleft to edit the code.">
    /**
    * Handles the HTTP <code>GET</code> method.
    * @param request servlet request
    * @param response servlet response
    */
    @Ov
    errid
    e
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
}
    /**
* Handles the HTTP <code>POST</code> method.
* @param request servlet request
* @param response servlet response
*/ @Override
```

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

    /**
    * Returns a short description of the servlet.
    */ @Override
    public String getServletInfo() {return "Short
        description";
    }

}
```

HOD,IT Dept.

Scheme Prepared by:
Mr. Shaik.Mabasha,
Dept. of IT, BEC.

Signature of Subject Teachers:

S.No	Name of the Faculty	Name of the college	Signature
1	B Krishnaih	BEC, IT Dept.	
2	K Sai Prasanth	BEC, IT Dept.	