14IT704

Hall Ticket Number:

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

November, 2019 **Information Technology**

Seventh Semester **Enterprise programming-II**

Time: Three Hours **Maximum :** 60 Marks

Answer Question No.1 compulsorily. (1X12 = 12 Marks)

Answer ONE question from each unit. (4X12=48 Marks)

1 Answer all questions (1X12=12 Marks)

a) How many variations are there in Java EE Applications.
There are web components and EJB Components. Servlets, JSPs, JSFs are web components.Session beans and Message driven Beans are EJB components

b) Differentiate doGet () and doPost () methods.
In doGet Method the parameters are appended to the URL and sent along with header information. In doPost, parameters are sent in separate line in the body. doGet () method generally is used to query or to get some information from the server. doPost () is generally used to update or post some information to the server.

c) What are dynamic web pages?
Web Pages that change its content based on the actions that are performed on the page.

d) List out Servlet API Packages?
RequestDispatcher
ServletConfig
ServletContext

e) What is web socket encoder?
An encoder takes a Java object and produces a representation that can be transmitted as a WebSocket message.

f) How many tags are provided in JSTL?
5 types of tags in JSTL
Core Tags
Function tags
Formatting tags
XML tags
SQL Tags

g) Differentiate Stateless and Stateful session beans.
Stateless session beans do not maintain state associated with any client. Each stateless session bean can server multiple clients. Stateful session beans maintain the state associated with a client. Each stateful session bean serves exactly one client.

h) List out any three annotations used in Stateful session beans
@stateful
@predestroy
@postconstruct

i) What is the Java Bean and Bean Development Kit (BDK)?
JavaBeans are classes that encapsulate many objects into a single object (the bean).
The Bean Developer Kit (BDK), is a simple example of a tool that enables you to create, configure, and connect a set of Beans.

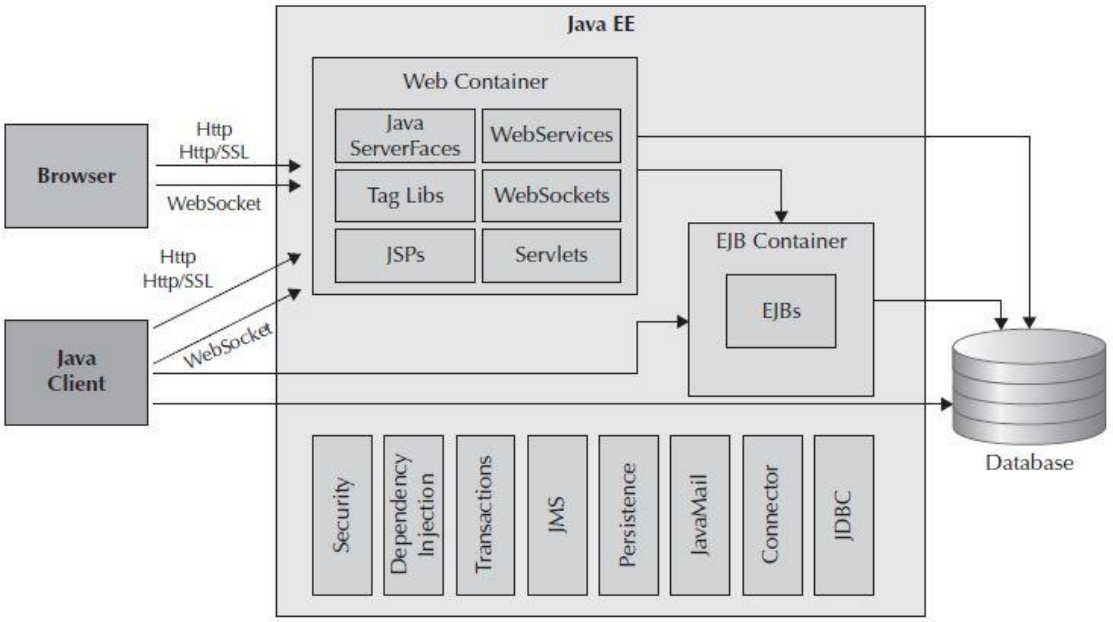
j) What are different components of WSDL?
A WSDL document has a definitions element that contains the other five elements, types, message, portType, binding and service

k) Can we maintain user session in web services?
Yes We can maintain user session.

l) Define SOAP and list out any two advantages.
SOAP is an XML-based protocol for accessing web services over HTTP. It has some specification which could be used across all applications.

UNIT I

2 a) Describe in detail about Architecture and goals of Java EE with neat diagram. 6M



In the diagram, we see that the large box, labeled Java EE, represents the Java EE platform. This refers to the runtime environment provided by a Java EE application server. All the Java EE code you write as a developer runs in this environment. A common term for this environment is the Java EE container, the word container derived from the idea that the environment envelops your application code.

The Java EE platform supports a wide variety of protocols that clients may use to interact with a Java EE application that it is running, the main client types being the browser client, which connects to the Java EE application using standard web protocols such as HTTP and WebSockets. Many Java EE applications have non-Java clients, perhaps a Java desktop application, or another Java EE application running on a different application server. This other client type can connect to a Java EE application running on the Java EE application server using standard web protocols, in addition to TCP-based protocols such as Remote Method Invocation (RMI).

The final part of the diagram is the boxes contained within the Java EE container box. These other boxes represent a variety of services that a Java EE application may choose to use. The security service enables a Java EE application to restrict access to its functions to only a certain set of known users. The dependency injection service enables a Java EE application to delegate the lifecycle management and discovery of some of its core components. The transaction service enables Java EE applications 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 a Java EE application to the ability to reliably send messages to other servers in the deployment environment of the Java EE application server. The Persistence service in the Java EE platform 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 a Java EE application 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 platform's main extensibility point comes with 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. Finally, 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.

b) Write short notes on Http Session Class with example program. 6M

The javax.servlet.http.HttpSession object is a representation of a series of interactions with a single web application from a single client. If a web application wishes to adapt its behavior for a particular user based on previous knowledge of that user, or based on the user's previous interactions with the web application, then the HttpSession is the object to use. Each HttpSession object associated with a particular client is available to all Java servlet instances, and it can be obtained each time the client makes a new HTTP request to a servlet by either of the methods

public HttpSession getSession()

Each HttpSession object has a identifier available by the getId() call that is unique across all the active sessions in the web container. HttpSessions ultimately expire, either by an explicit call to its
public void invalidate()
method, or because the session times out because the client that the session is representing doesn't make any calls to the web application, defined by a timeout quantity controlled by the methods
public int getMaxInactiveInterval()
public void setMaxInactiveInterval(int interval)
These methods allow the web container the opportunity to invalidate the sessions once the maximum period of inactivity has been reached. This does not mean that the client has not been accessing other web applications deployed to the web container: like the ServletContext instances, HttpSession instances cannot cross the boundaries of web applications.

```
java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Session extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
HttpSession session = request.getSession(true);
Date created = new Date(session.getCreationTime());
Date accessed = new Date(session.getLastAccessedTime());
out.println("ID " + session.getId()+"<br>");
out.println("Created: " + created+"<br>");
out.println("Last Accessed: " + accessed+"<br>");
String dataName = request.getParameter("dataname");
if (dataName != null && dataName.length() > 0) {
String dataValue = request.getParameter("datavalue");
session.setAttribute(dataName, dataValue);
}
Enumeration e = session.getAttributeNames();
while (e.hasMoreElements()) {
String name = (String)e.nextElement();
String value = session.getAttribute(name).toString();
out.println("<br>"+name + " = " + value);
}
}
}
```

Session.html

```
<html>
<head>
<title>Sessions Example</title>
</head>
<body>
<h1> An example for session attributes </h1>
<form action="Session" method=GET>
Name of Session Attribute:
<input type=text size=20 name=dataname>
<br>
Value of Session Attribute:
<input type=text size=20 name=datavalue>
<br>
<input type=submit>
</form>
</body>
</html>
```

(OR)

3 a) What is difference between Servlet Config and Servlet Context? 6M

Servlet Config	Servlet Context
Servlet config object represent single servlet	It represent whole web application running on particular JVM and common for all the servlet
Its like local parameter associated with particular servlet	Its like global parameter associated with whole application
It's a name value pair defined inside the servlet section of web.xml file so it has servlet wide scope	`ServletContext` has application wide scope so define outside of servlet tag in web.xml file.
`getServletConfig()` method is used to get the config object	`getServletContext()` method is used to get the context object.
for example shopping cart of a user is a specific to particular user so here we can use servlet config	To get the MIME type of a file or application session related information is stored using servlet context object.

b) Write servlet program for retrieving the http parameter values and display it in next screen? 6M

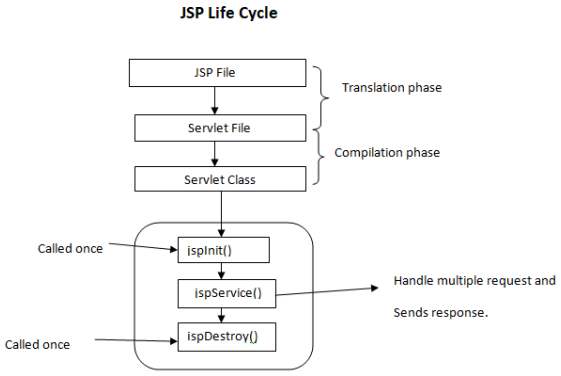
Index.html

```
<html>
   <head>
      <title>Http Parameters</title>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
   </head>
   <body>
      <form action="NewServlet" method="get">
         Enter Name:<input name="t1" type="text" />
         <input type="submit"/>
      </form>
   </body>
</html>
```

NewServlet.java

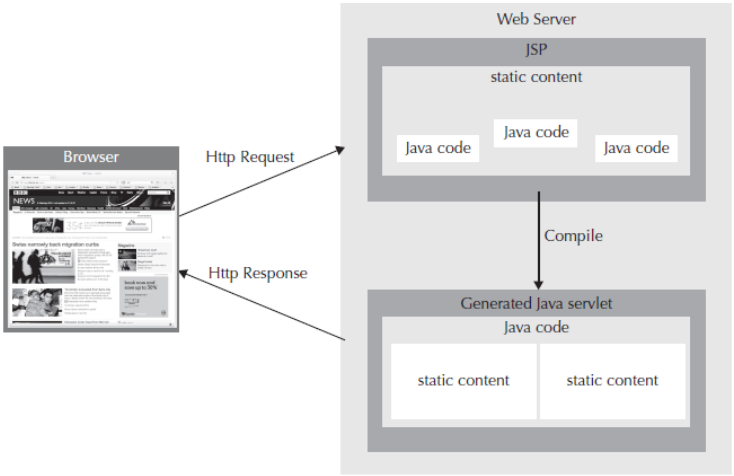
```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author ravi
 */
@WebServlet(urlPatterns = {"/NewServlet"})
public class NewServlet extends HttpServlet {

   /**
    * Processes requests for both HTTP <code>GET</code> and <code>POST</code>
    * methods.
    *
    * @param request servlet request
    * @param response servlet response
    * @throws ServletException if a servlet-specific error occurs
    * @throws IOException if an I/O error occurs
    */
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
     PrintWriter out=response.getWriter();
     String name=request.getParameter("t1");
         out.println("<h1>Welcome " + name + "</h1>");

   }
```

4 a) Describe JSP life cycle with neat diagram. 6M



Method	Description
jspInit()	This method is called from the init() method and it can be overridden
jspDestroy()	This method is called from the servlets destroy() method and it too can be overridden
_jspService()	This method is called from the servlets service() method which means it runs in a separate thread for each request, the container passes the request and response object to this method. You cannot override this method.

b) Explain the methodologies that can be implemented to create dynamic web pages. Discuss architecture of any one? 6M



A JavaServer Page (JSP) is a Java servlet turned inside out. In other words, a JSP is equivalent to a servlet in the sense that almost everything you can do in a Java servlet you can also do in a JSP. JSP programming, instead of the static content of the web component being embedded inside Java code, the dynamic fragments of the JSP are embedded as Java code within the static content.
This feature makes JSPs especially well suited to the creation of web components that contain a great deal of static content that is frequently tweaked and adjusted in a publishing tool, together with small amounts of embedded dynamic content. In other words, JSPs are well suited to creating content for a typical dynamic HTML website. In such sites, many of the pages are static: static layouts; infrequently changing portions of static content that embeds more dynamic generated content, such as breaking news headlines; personalized information such as account names; and dynamically retrieved data such as top news headlines or recent purchases.

(OR)

5 a) Write a short note on Java Server Faces Tags? 6M

Component	Tag	Example
Button	`<h:commandButton>`	`<h:commandButton value="Submit" action="#{bookingBean.submit}" />`
Hyperlink	`<h:commandLink>`	`<h:commandLink value="Learn more..." action="learn.xhtml" />`
Table	`<h:dataTable>` `<h:column>`	`<h:dataTable value="#{widgetBean.items}" var="item"> <h:column> #{item.name} </h:column> <h:column> #{item.age} </h:column> </h:dataTable>`
Form	`<h:form>`	`<h:form> <h:commandButton value="Submit" action="#{bookingBean.submit}" /> </h:form>`
Image	`<h:graphicImage>`	`<h:graphicImage value="#{widget.imageUri}" />`
Text input	`<h:inputText>` `<h:inputArea>`	`<h:inputText value ="#{widget.name}" />`
Component messages	`<h:messages>`	`<h:messages errorStyle="color:red" />`
Check box	`<h:selectBooleanCheckbox>`	`<h:selectBooleanCheckbox value="#{widgetBean.working}" />`
List, single selection	`<h:selectOneMenu>`	`<h:selectOneMenu value="#{widget.color}"> <f:selectItem itemValue="red" itemLabel="Red" />`

b) Create a Web Application to demonstrate Web Sockets. 6M

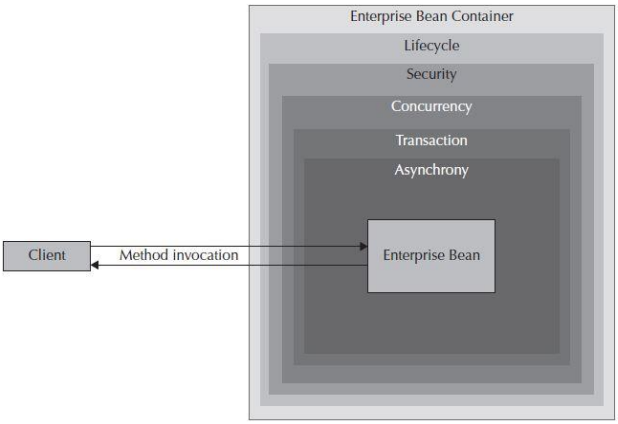
```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Web Socket Clock</title>
<script language="javascript" type="text/javascript">
var websocket;
var last_time;
function init() {
output = document.getElementById("output");
}
function start_clock() {
var wsUri = "ws://localhost:8080/clock-app/clock";
websocket = new WebSocket(wsUri);
websocket.onmessage = function (evt) {
last_time = evt.data;
writeToScreen("<span style='color: blue;'>" + last_time + "</span>");
};
websocket.onerror = function (evt) {
writeToScreen('<span style="color: red;">ERROR:</span> ' + evt.data);
websocket.close();
```

```
};
}
function stop_clock() {
websocket.send("stop");
}
function writeToScreen(message) {
var pre = document.createElement("p");
pre.style.wordWrap = "break-word";
pre.innerHTML = message;
oldChild = output.firstChild;
if ( oldChild == null) {
output.appendChild(pre);
} else {
output.removeChild(oldChild);
output.appendChild(pre);
}
}
window.addEventListener("load", init, false);
</script>
</head>
<body>
<div style="text-align: center;font-family: Arial; font-size: large">
WebSocket Clock
<br></br>
<form action="">
<input
onclick="start_clock()"
title="Press to start the clock on the server"
value="Start"
type="button">
<input
onclick="stop_clock()"
title="Press to stop the clock on the server"
value="Stop"
type="button">
</form>
<div id="output"></div>
</div>
</body>
</html>
```

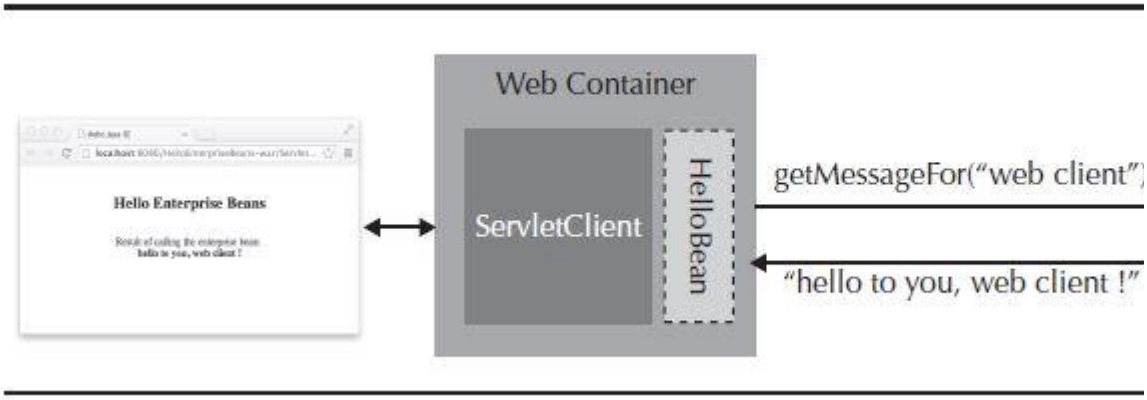
UNIT III

6 a) Diagrammatically illustrate and discuss the Enterprise Java Bean(EJB) architecture? 6M

Enterprise Bean architecture:



This example is a Java EE application containing a single Enterprise Bean, which is called by a Java servlet. When you run the application, the web browser calls the Java servlet. You will see something like this shown in Figure.



There are three Java classes in this application: the Java servlet, called ServletClient, and two Java classes that make up the Enterprise Bean: `HelloBean` and `HelloBeanImpl`.

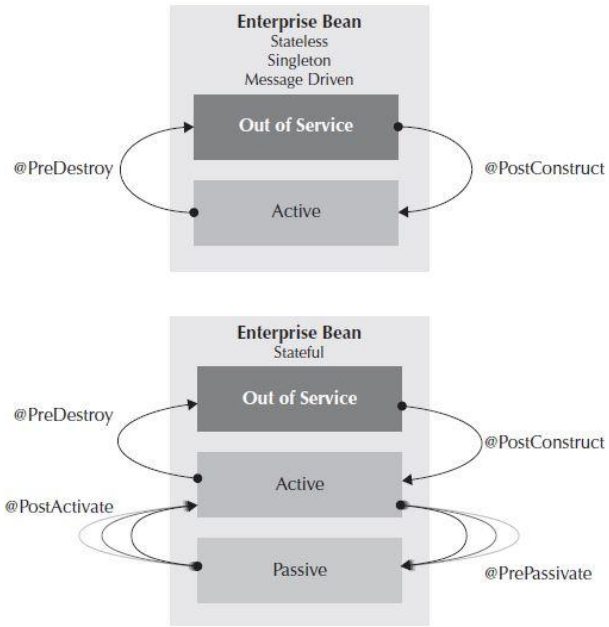
The Java class is actually an interface. It defines a single method, and other than the `@Remote` annotation at the class level, it is just like an ordinary Java interface. What the `@Remote` annotation does is turn this interface into an Enterprise Bean remote interface, which is to say, this is an interface that is used by an Enterprise Bean to publish its methods to all its potential clients, whether they be application clients, web components, or other Enterprise Beans.

Enterprise Bean implementation class implements the remote interface. The only extra thing over a regular Java class that implements an interface is the use of the @Stateful annotation from the javax.ejb package. This marks the implementation class as a particular kind of session bean called a stateful session bean. In short, the Enterprise Bean container creates exactly one instance of this kind of bean for each client that makes calls to it.

6M

(OR)

7 a) Draw the life cycle of entity bean and Explain it? 6M

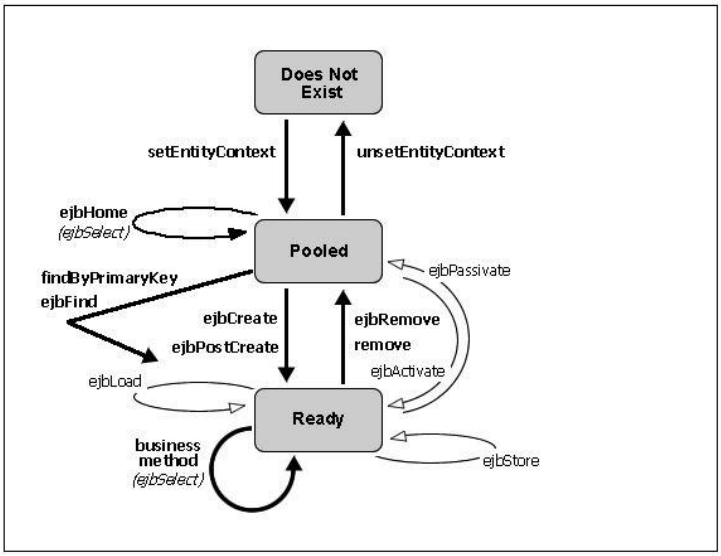


the life cycle of an entity bean. An entity bean has the following three states:

- **Does not exist**. In this state, the bean instance simply does not exist.
- **Pooled state** . When WebLogic server is first started, several bean instances are created and placed in the pool. A bean instance in the pooled state is not tied to particular data, that is, it does not correspond to a

record in a database table. Additional bean instances can be added to the pool as needed, and a maximum number of instances can be set (for more information, see the @Entity Annotation).

- **Ready state**. A bean instance in the ready state is tied to particular data, that is, it represents an instance of an actual business object.



b) Explain about Java Persistence Query Language? 6M

Java Persistence Query Language 2M for explanation 2M for example queries

The Java Persistence API defines an SQL-like query language that can be used to formulate queries against persistent data managed by an EntityManager.

JPQL is heavily inspired by SQL, and anyone already familiar with a SQL variant will find JPQL very easy to use. The basic statements are SELECT, UPDATE, and DELETE. For example,

SELECT a.firstName FROM Author a WHERE a.lastName = "Coward" ORDER BY a.firstName

would return all the first names of authors in alphabetical order from the Author table with last name Coward.

DELETE from Author WHERE lastName = "James"

would delete all authors from the Author table where the last name is James.

Most pertinently to JPA applications, JPQL statements may contain named parameters. A named parameter in a JPQL statement always begins with a colon.

JPA queries are created with the EntityManager. There are two primary ways this can be done:

either dynamically using the createQuery() API, for example

Query q = myEntityManager.createQuery("SELECT a.firstName,a.lastName FROM Author a ORDER BY a.lastName");

or by using 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;
...
}
```

UNIT IV

8 a) How is XML used in SOAP? 6M

XML

Because XML is the perfect integration technology that solves the problem of data independence and interoperability, it is the DNA of SOAP web services. It is used not only as the message format but also as the way the services are defined (WSDL) or exchanged (SOAP). Associated with these XML documents, schemas (XSD) are used to validate the data exchanged between the consumer and the provider. Historically, SOAP web services evolved from the basic idea of "RPC (Remote Procedure Call) using XML."

XML is used to write both SOAP messages and to define the services in WSDL file the example below demonstrates how a SOAP message can be written in XML for a credit card application.

SOAP Request file:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:cc="http://chapter14.javaee7.book.agoncal.org/">
<soap:Header/>
<soap:Body>
<cc:validate>
<arg0 number="123456789011" expiry_date="10/12"
control_number="544" type="Visa"/>
</cc:validate>
</soap:Body>
</soap:Envelope>
```

SOAP Response file:

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" □
xmlns:cc="http://chapter14.javaee7.book.agoncal.org/">
<soap:Body>
<cc:validateResponse>
<return>true</return>
</cc:validateResponse>
</soap:Body>
</soap:Envelope>
```

b) Write an application to create SOAP Web service. 6M

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();

        } finally {
            out.close();
        }
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on the
left to edit the code.">
    /**
    * Handles the HTTP <code>GET</code> method.
    * @param request servlet request
    * @param response servlet response
    */
    @Override
    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";
    }

}
```

(OR)

9 a) What are the types of information included in SOAP header and explain it 6M

SOAP Header element contains application-specific information (like authentication, payment, etc) about the SOAP message. If the Header element is present, it must be the first child element of the Envelope element.

The mustUnderstand Attribute 2M

The SOAP mustUnderstand attribute can be used to indicate whether a header entry is mandatory or optional for the recipient to process.

If you add mustUnderstand="1" to a child element of the Header element it indicates that the receiver processing the Header must recognize the element. If the receiver does not recognize the element it will fail when processing the Header.

The actor Attribute 2M

A SOAP message may travel from a sender to a receiver by passing different endpoints along the message path. However, not all parts of a SOAP message may be intended for the ultimate endpoint, instead, it may be intended for one or more of the endpoints on the message path.

encodingStyle 2M

The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and it will apply to that element's contents and all child elements.

b) Write WSDL file for credit card validation service? 6M

WSDL file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions □
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" □
xmlns:tns="http://chapter14.javaee7.book.agoncal.org/" □
xmlns:xsd="http://www.w3.org/2001/XMLSchema" □
xmlns="http://schemas.xmlsoap.org/wsdl/" □
targetNamespace="http://chapter14.javaee7.book.agoncal.org/"
name="ValidatorService">
<types>
<xsd:schema>
<xsd:import
namespace="http://chapter14.javaee7.book.agoncal.org/" □
schemaLocation="http://localhost:8080/chapter14/ValidatorSer
vice?xsd=1"/>
</xsd:schema>
</types>
```

```
<message name="ValidateCreditCard">
<part name="Credit-Card" type="tns:creditCard"/>
</message>
<message name="ValidateCreditCardResponse">
<part name="IsValid" type="xsd:boolean"/>
</message>
<message name="ValidateCreditCardNumber">
<part name="Credit-Card-Number" type="xsd:string"/>
</message>
<message name="ValidateCreditCardNumberResponse"/>
<portType name="CardValidator">
<operation name="ValidateCreditCard">
<input message="tns:ValidateCreditCard"/>
<output message="tns:ValidateCreditCardResponse"/>
</operation>
<operation name="ValidateCreditCardNumber">
<input message="tns:ValidateCreditCardNumber"/>
<output message="tns:ValidateCreditCardNumberResponse"/>
</operation>
</portType>
<binding                    name="CreditCardValidatorBinding"
type="tns:CardValidator">
<soap:binding
transport="http://schemas.xmlsoap.org/soap/http"
style="rpc"/>
<operation name="ValidateCreditCard">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
<operation name="ValidateCreditCardNumber">
<soap:operation soapAction=""/>
<input>
<soap:body use="literal"/>
</input>
<output>
<soap:body use="literal"/>
</output>
</operation>
</binding>
<service name="ValidatorService">
<port                              name="CreditCardValidator"
binding="tns:CreditCardValidatorBinding">
<soap:address
location="http://localhost:8080/chapter14/ValidatorService"/
>
</port>
</service>
</definitions>
```

Scheme prepared by
Mr. K Sai Prasanth
Department of IT

Head of the Department

Paper Evaluators:

S.no	Name of Examiner	Name of College	Signature