Q.1. Implement award list of MCA 1st year as a XML document. The table must have semester No, student's Enrollment No, TEE marks of all the theory subjects, practical subjects and assignments.

Q. 2. Create a student enquiry system using EJB which a student could enquire for taking admission to BCA & MCA programmes of the University. More fields may be added if required.

Ans: A sample EJB application for student enquiry system Course Remote Interface:

```
import javax.ejb.*;
import java.rmi.RemoteException;
public interface Course extends EJBObject
{
public String getCourseUniversity() throws RemoteException;
public void setCourseUniversity(String CourseUniversity) throws RemoteException;
public String getCourseName() throws RemoteException;
public void setCourseName(String coursename) throws RemoteException;
public String getCourseDuration() throws RemoteException;
public void setCourseDuration(String courseduration) throws RemoteException;
public String getCourseType() throws RemoteException;
public void setCourseType(String coursetype) throws RemoteException;
public String getCourseCost() throws RemoteException;
public void setCourseCost(String coursecost) throws RemoteException;
}
```

Course Home Interface

CODE

```
import javax.ejb.*;
import java.rmi.RemoteException;
```

```
public interface CourseHome extends EJBHome
{
public Course create(String CourseUniversity) throws
RemoteException,CreateException;
public Course findByPrimaryKey(String CourseUniversity) throws
RemoteException,FinderException;
}
```

Course Entity Bean

CODE

```
import javax.ejb.*;
import java.rmi.RemoteException;
import java.sql.*;
public interface CourseBean extends EntityBean
{
transient private EntityContext ejbContext;
public String CourseUniversity;
public String courseName;
public String courseDuration;
public String courseType;
public String courseCost;
public String getCourseUniversity()
{
return CourseUniversity;
}
public void setCourseUniversity(String school)
{
CourseUniversity=school;
}
```

```
public String getCourseName()
{
return courseName;
}
public void setCourseName(String name)
{

courseName=name;
}
public String getCourseDuration()
{
return courseDuration;
}
public void setCourseDuration(String duration)
{
courseDuration=duration;
}
public String getCourseType()
{
return courseType;
}
public void setCourseType(String type)
{
courseType=type;
}
public String getCourseCost()
{
return courseCost;
}
```

```
public void setCourseCost(String cost)
{
courseCost=cost;
}
public Course ejbCreate(String school)
{
CourseUniversity=school;
return null;
}
public void ejbPostCreate(String school)
{}
public void setEntityContext(EntityContext ctx)
{
ejbContext=ctx;
}
public void unsetEntityContext()
{
ejbContext=null;
}
public void ejbActivate(){}
public void ejbPassivate(){}

public void ejbLoad()
{
Connection con;
try
{
String primaryKey=(String)ejbContext.getPrimaryKey();
con=this.getConnection();
```

```
Statement stm=con.createStatement(SELECT * FROM Course
WHERE
CourseUniversity="+primaryKey);
ResultSet rst=stm.executeQuery();
if(rst.next())
{
CourseUniversity=rst.getString("CourseUniversity");
courseName=rst.getString("CourseName");
courseDuration=rst.getString("CourseDuration");
courseType=rst.getString("CourseType");
courseCost=rst.getString("CourseCost");
}
}
catch(SQLException sqle)
{
throw new EJBException(sqle);
}
finally
{
if(connection !=null)
con.close();
}
}
```

Q.3. Create a database of employee working in the university. Write programme using JSP & JDBC to display names of those employees who are pursuing academic programme from different schools of the University.

Ans:-

DATABASE

Database name:**Employee**

```
CREATE TABLE employee
(
empID VARCHAR(10)NOT NULL ,
emp_name VARCHAR(30),
emp_enrol VARCHAR(9),
emp_address VARCHAR(70),
emp_course VARCHAR(10),
CentreID VARCHAR(4),
PRIMARY KEY(StudentID)
);
CREATE TABLE emp_schools
(
SchoolsID INT NOT NULL AUTO_INCREMENT,
Schools_name VARCHAR(30),
Schools_address VARCHAR(70),
empID VARCHAR(10),
PRIMARY KEY(SchoolsID),
FOREIGN KEY(empID) REFERENCES Employee (empID)
);
```

JSP page for displaying Employee working in Universiy

```
<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN">
```

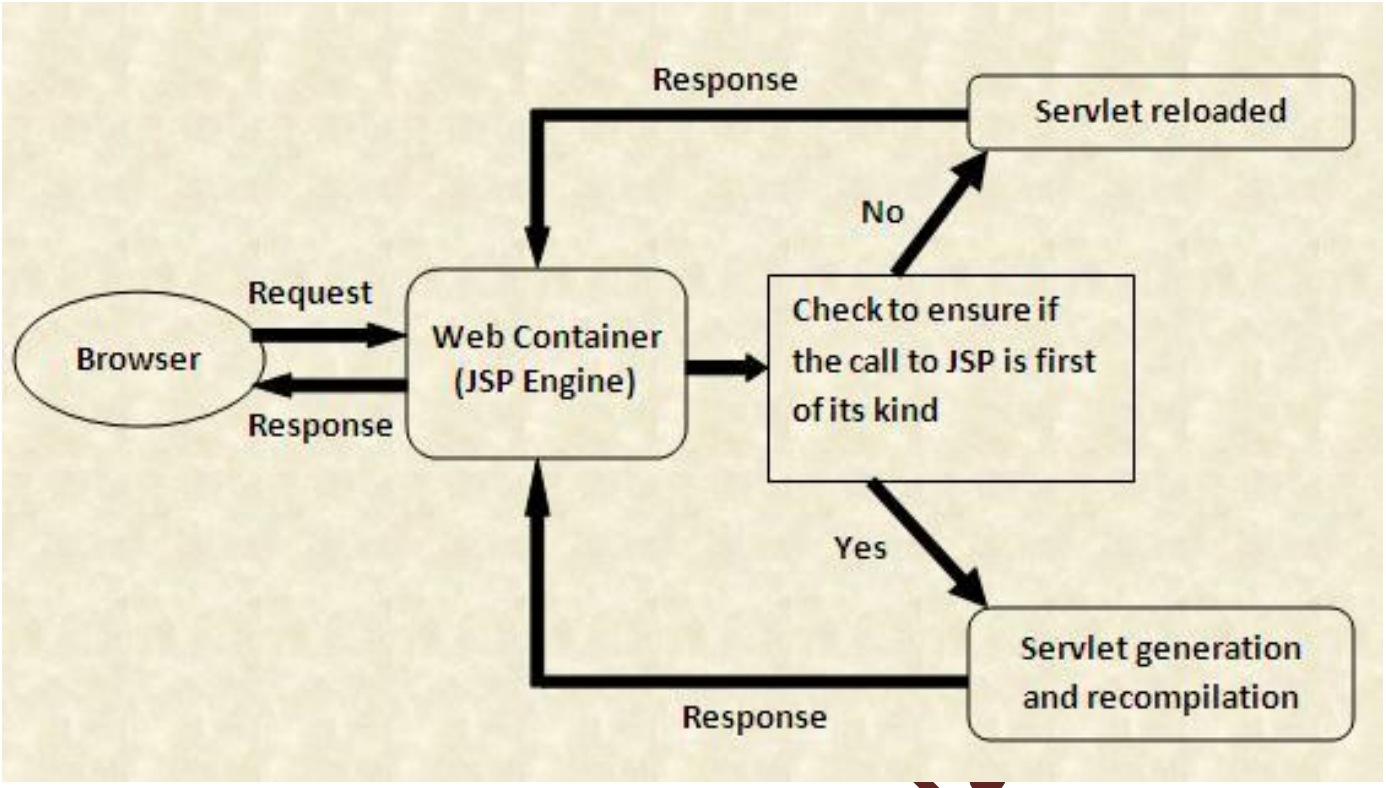
```
<html>
<head>
<title>BPO Working student</title>
</head>
<body bgcolor=#ffccdd text=#111ccc>
<h3>Information of student who are currently working in BPO Company</h3>
<table cellspacing="5" cellpadding="5" border="1" align="center"width="70%">
<tr>
<td align="right">Name:</td>
<td align="right">Address:</td>
<td align="right">Schools of University:</td>
</tr>
<%@page language="java"%>
%@page import="java.sql.*"%

con = null;
Statement stm = null;
try
{
Class.forName("org.gjt.mm.mysql.Driver");
String dbURL="jdbc:mysql://localhost/University";
String username="root";
String password="";
con=DriverManager.getConnection(dbURL,username,password);
stm=con.createStatement();
ResultSet rs=stm.executeQuery("SELECT
Employee.emp_name,employee.emp_address, emp_schools. Schools_name
FROM emp, emp_schools WHERE
emp.empID= emp_schools.empID");
```

```
while(rs.next()){%>
<TR>
<TD><%out.println(rs.getString("Emp_name"));%></TD>
<TD><%out.println(rs.getString("emp_address"));%></TD>
<TD><%out.println(rs.getString("schools_name"));%></TD>
</TR>
<%}%>
<%
rs.close();
stm.close();
con.close();
}catch(Exception e){e.printStackTrace();}%>
</table>
</body>
</html>
```

Q.4. Write a code in JSP to stop the caching of a page by a browser.

Ans:JavaServer Pages — JSP is a Java based technology that is used to develop dynamic web sites. With JSP, web designers and developers can quickly incorporate dynamic elements into web pages using embedded Java and markup tags. These tags provide the HTML designer with a way to access data and business logic stored inside Java objects.




```
<%
  response.setHeader( "Pragma", "no-cache" );
  response.setHeader( "Cache-Control", "no-cache" );
  response.setDateHeader( "Expires", 0 );
%>
```

The same effect can be achieved by using meta tags in the HTML header:

```
<meta http-equiv="Pragma" content="no-cache">
<meta http-equiv="Cache-Control" content="no-cache">
<meta http-equiv="Expires" content="Sat, 01 Dec 2001 00:00:00 GMT">
```

The Cache-Control header was added in HTTP 1.1, while the other two were also present in HTTP 1.0

Q.5. Write code to use a form to authenticate a client using the session information stored in the HTTP session object to the web server.

Ans:- **Creating a Web Client for Form-Based Authentication:** The web client in this example is a standard JSP page, and annotations are not used in JSP pages because JSP pages are compiled as they are presented to the browser.

Creating the Login Form and the Error Page

The login page can be an HTML page, a JSP page, or a servlet, and it must return an HTML page containing a form that conforms to specific naming conventions (see the Java Servlet 2.5 specification for more information on these requirements). To do this, include the elements that accept user name and password information between `<form></form>` tags in your login page. The content of an HTML page, JSP page, or servlet for a login page should be coded as follows:

```
<form method=post action="j_security_check" >
    <input type="text"  name= "j_username" >
    <input type="password"  name= "j_password" >
</form>
```

The full code for the login page used in this example can be found at *tut-install*`/javaeetutorial5/examples/web/hello1_formauth/web/logon.jsp`. An example of the running login

form page is shown later in Figure 30–6. Here is the code for this page:

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

```
<h2>Hello, please log in:</h2>
<br><br>
<form action="j_security_check" method=post>
    <p><strong>Please Enter Your User Name: </strong>
    <input type="text" name="j_username" size="25">
    <p><p><strong>Please Enter Your Password: </strong>
    <input type="password" size="15" name="j_password">
    <p><p>
    <input type="submit" value="Submit">
    <input type="reset" value="Reset">
</form>
</html>
```

The login error page is displayed if the user enters a user name and password combination that is not authorized to access the protected URI. For this example, the login error page can be found at *tut-install*`/javaeetutorial5/examples/web/hello1_formauth/web/logonError.jsp`. For this example, the
login error page explains the reason for receiving the error page and provides a link that will allow the user to try again. Here is the code for this page:

```
<html>
<head>
    <title>Login Error</title>
</head>
<body>
```

```
    <c:url var="url" value="/index.jsp"/>
    <h2>Invalid user name or password.</h2>

    <p>Please enter a user name or password that is authorized
to access this
    application. For this application, this means a user that
has been created in the
    <code>file</code> realm and has been assigned to the
<em>group</em> of
    <code>user</code>.  Click here to <a href="${url}">Try
Again</a></p>
</body>
</html>
```

Specifying a Security Constraint

This example takes a very simple JSP page-based web application and adds form-based security to this application. The JSP page is exactly the same as the JSP page used in the example described in Web Modules. All security for this example is declared in the deployment descriptor for the application. A security constraint is defined in the deployment descriptor that tells the server to send a login form to collect user data, verify that the user is authorized to access the application, and, if so, display the JSP page to the user.

If this client were a web service endpoint and not a JSP page, you could use annotations to declare security roles and to specify which roles were allowed access to which methods. However, there is no resource injection in JSP pages, so you cannot use annotations and must use the equivalent deployment descriptor elements.

Deployment descriptor elements are described in Declaring Security Requirements in a Deployment Descriptor.

The following sample code shows the deployment descriptor used in this example of form-based login authentication, which can be found in *tut-install*/javaeetutorial5/examples/web/hello1_formauth/web/WEB-INF/web.xml.

```
<!-- FORM-BASED LOGIN AUTHENTICATION EXAMPLE -->
<?xml version="1.0" encoding="UTF-8"?>
 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
version="2.5"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
     http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

    <display-name>hello1_formauth</display-name>
       <servlet>
            <display-name>index</display-name>
            <servlet-name>index</servlet-name>
            <jsp-file>/index.jsp</jsp-file>
       </servlet>
     <security-constraint>
            <display-name>SecurityConstraint</display-name>
           <web-resource-collection>
                <web-resource-name>WRCollection</web-
resource-name>
```

```
                <url-pattern>/*</url-pattern>
            </web-resource-collection>
               <auth-constraint>
                     <role-name>loginUser</role-name>
               </auth-constraint>
               <user-data-constraint>
                   <transport-guarantee>NONE</transport-guarantee>
               </user-data-constraint>
         </security-constraint>
        <login-config>
               <auth-method>FORM</auth-method>
           <form-login-config>
                    <form-login-page>/logon.jsp</form-login-page>
                   <form-error-page>/logonError.jsp</form-error-
page>
            </form-login-config>
      </login-config>
      <security-role>
          <role-name>loginUser</role-name>
     </security-role>
</web-app>
```

Q.6 Write a web based student registration application where the students can register online with their enrollment no. You are required to use JSP, Servelet and JDBC.

Ans:

StudentRegistration

index.html

login.html

WEB-INF

web.xml

classes

login.class

login.java

regform.class

regform.java

Index.html

```
<html>
<head>
<title>registration</title>
</head>
<body>
<form method="post" action="reg">
NAME:<input type="text"name="t1"/><br>
ENROLLMENT:<input type="text"name="t2"/><br>
PASSWORD:<input type="password"name="t3"/><br>
<input type="submit"value="send"/><br>
```

```
</form>
<a href='login.html'>if already registered plz login</a>
</body>
</html>
```

login.html

```
<html>
<head>
<title>login</title>
</head>
<body>
<form method="post" action="login">
  <h2>ENTER VALID USER NAME AND PASSWORD</h2>
  <p><strong>USER NAME:</strong>
<input type="text"name="t1"/><br>
    <strong>PASSWORD:</strong>
    <input type="password"name="t2"/><br>
  <input type="submit"value="Login"/><br>
  </p>
</form>
<a href='index.html'>new user create account</a>
</body>
</html>
```


web.xml

```
<web-app>
 <servlet>
  <servlet-name>s1</servlet-name>
```

```
 <servlet-class>regform</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>s1</servlet-name>
 <url-pattern>/reg</url-pattern>
</servlet-mapping>

<servlet>
 <servlet-name>s2</servlet-name>
 <servlet-class>login</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>s2</servlet-name>
 <url-pattern>/login</url-pattern>
</servlet-mapping>
</web-app>
```

regform.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class regform extends HttpServlet{
 public void doPost(HttpServletRequest req,HttpServletResponse res)throws
IOException,ServletException{
  res.setContentType("text/html");
  PrintWriter out=res.getWriter();
  String n=req.getParameter("t1");
```

```
String e=req.getParameter("t2");
String p=req.getParameter("t3");
try{
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con=DriverManager.getConnection("jdbc:odbc:school");
 PreparedStatement st=con.prepareStatement("insert into student values(?,?,?)");
 st.clearParameters();
 st.setString(1,n);
 st.setInt(2,Integer.parseInt(e));
 st.setString(3,p);
 st.executeUpdate();
 con.close();

}catch(Exception ex){
 ex.printStackTrace(System.out);
}
out.write("ur account has been created, <a href='login.html'>u can login now</a>");

}
}
```

login.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class login extends HttpServlet{
```

```
 public void doPost(HttpServletRequest req,HttpServletResponse res)throws
IOException,ServletException{
  res.setContentType("text/html");
  PrintWriter out=res.getWriter();
  String n=req.getParameter("t1");
  String p=req.getParameter("t2");

  try{
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   Connection con=DriverManager.getConnection("jdbc:odbc:school");
   PreparedStatement st=con.prepareStatement("select * from student where name=? and
pass=?");
   st.clearParameters();
   st.setString(1,n);
   st.setString(2,p);
   ResultSet rs=st.executeQuery();
   boolean b=rs.next();
   if(b==true){
    out.write("WELCOME");

   }
   else{
    out.write("Login failed <a href='login.html'>TRY AGAIN</a>");
   }
   con.close();

  }catch(Exception ex){
```

```
  ex.printStackTrace(System.out);
 }
 }
}
```

Q.7. Differentiate between structured, semi-structured and unstructured data.

Ans.: Data that resides in a fixed field within a record or file is called structured data. This includes data contained in relational databases and spreadsheets.

Structured data first depends on creating a data model – a model of the types of business data that will be recorded and how they will be stored, processed and accessed. This includes defining what fields of data will be stored and how that data will be stored: data type (numeric, currency, alphabetic, name, date, address) and any restrictions on the data input (number of characters; restricted to certain terms such as Mr., Ms. or Dr.; M or F).

Structured data has the advantage of being easily entered, stored, queried and analyzed. At one time, because of the high cost and performance limitations of storage, memory and processing, relational databases and spreadsheets using structured data were the only way to effectively manage data. Anything that couldn't fit into a tightly organized structure would have to be stored on paper in a filing cabinet.

- data is organised in semantic chunks (entities)
- similar entities are grouped together (relations or classes)
- entities in the same group have the same descriptions (attributes)
- descriptions for all entities in a group (schema)
- have the same defined format
- have a predefined length
- are all present
- and follow the same order

Semi-structured-Semi-structured data is data that is neither raw data, nor typed data in a conventional database system. It is structured data, but it is not organized in a rational model, like a table or an object-based graph. A lot of data found on the Web can be described as semi-structured. Data integration especially makes use of semi-structured data.

Semi-structured data is a form of structured data that does not conform with the formal structure of data models associated with relational databases or other forms of data tables, but nonetheless contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data. Therefore, it is also known as self-describing structure.

- idea predates XML but not HTML
- data is available electronically in
- database systems
- file systems, e.g., bibliographic data, Web data
- data exchange formats, e.g., EDI, scientific data
- attempt to reconcile database and document "worlds"
- semi-structured data
    - organised in semantic entities
    - similar entities are grouped together
    - entities in same group may not have same attributes
    - order of attributes not necessarily important
    - not all attributes may be required
    - size of same attributes in a group may differ
    - type of same attributes in a group may differ

Unstructured Data -Unstructured data is a generic label for describing any corporate information that is not in a database. Unstructured data can be textual or non-textual. Textual unstructured data is generated in media like email messages, PowerPoint presentations, Word

documents, collaboration software and instant messages. Non-textual unstructured data is generated in media likeJPEG images, MP3 audio files and Flash video files.

Unstructured Data (or unstructured information) refers to information that either does not have a pre-defined data model or is not organized in a pre-defined manner. Unstructured information is typically text-heavy, but may contain data such as dates, numbers, and facts as well. This results in irregularities and ambiguitiesthat make it difficult to understand using traditional computer programs as compared to data stored in fielded form in databases or annotated (semantically tagged) in documents.

- data can be of any type
- not necessarily following any format or sequence
- does not follow any rules
- is not predictable
- examples include
    - text
    - video
    - sound
    - images

Q.8. Describe the use of SSL authentication in Java clients with the help of a sample code.

Ans.- **SSL Authentication:** Discuss SSL Authentication that takes place between the Application layer and the TCP/IP layer. The SSL protocol operates between the application layer and the TCP/IP layer. This allows it to encrypt the data stream itself, which can then be transmitted securely, using any of the application layer protocols. In this both symmetric and asymmetric encryption is used.

SSL uses certificates for authentication — these are digitally signed documents which bind the public key to the identity of the private key owner. Authentication happens at connection time, and is independent of the application or the application protocol.

Certificates are used to authenticate clients to servers, and servers to clients; the mechanism used is essentially the same in both cases. However, the server certificate is mandatory — that is, the server must send its certificate to the client — but the client certificate is optional: some clients may not support client certificates; other may not have certificates installed. Servers can decide whether to require client authentication for a connection.

Using SSL Authentication in Java Clients:

JSSE (Java Secure Socket Extesnsion) and Web Logic Server

uses is static to the server configuration and is not replaceable by user applications. You cannot plug different JSSE implementations into WebLogic Server to have it use those implementations for SSL.

- The WebLogic implementation of JSSE does support JCE Cryptographic Service Providers (CSPs), however, due to the inconsistent provider support for JCE, BEA cannot guarantee that untested providers will work out of the box. BEA has tested WebLogic Server with the following providers:

  i) The default JCE provider (SunJCE provider) that is included with JDK

  ii) The nCipher JCE provider.

WebLogic Server uses the HTTPS port for SSL. Only SSL can be used on that port. SSL encrypts the data transmitted between the client and WebLogic Server so that the username and password do not flow in clear text.

JSSE is a set of packages that support and implement the SSL and TLS v1 protocols, making those capabilities available. BEA WebLogic Server provides Secure Sockets Layer (SSL) support for encrypting data transmitted between Web Logic Server clients and servers, Java clients, Web browsers, and other servers. Web Logic Server's Java Secure Socket Extension (JSSE) implementation can be used by WebLogic clients. Other JSSE implementations can be used for their client-side code outside the server as well.

The following restrictions apply when using SSL in WebLogic server-side applications:

- The use of third-party JSSE implementations to develop WebLogic server applications is not supported. The SSL implementation that WebLogic Server

Using JNDI Authentication:

Java clients use the Java Naming and Directory Interface (JNDI) to pass on credentials to the WebLogic Server. A Java client establishes a connection with WebLogic Server by getting a JNDI InitialContext. The Java client then, uses the InitialContext to look up the resources it needs in the WebLogic Server JNDI tree.

Example1.0: Example of One-Way SSL Authentication Using JNDI

```
...
Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY,
          "weblogic.jndi.WLInitialContextFactory");
    env.put(Context.PROVIDER_URL, "t3s://weblogic:7002");
env.put(Context.SECURITY_PRINCIPAL, "javaclient");
env.put(Context.SECURITY_CREDENTIALS, "javaclientpassword");
ctx = new InitialContext(env);
```

Writing Applications that Use SSL

- Communicating Securely From WebLogic Server to Other WebLogic Servers
- Writing SSL Clients
- Using Two-Way SSL Authentication
- Two-Way SSL Authentication with JNDI
- Using a Custom Host Name Verifier
- Using a Trust Manager
- Using an SSLContext
- Using an SSL Server Socket Factory
- Using URLs to Make Outbound SSL Connections

Writing SSL Clients

This section describes, by way of example, how to write various types of SSL clients. Examples of the following types of SSL clients are provided:

- SSLClient
- SSLSocketClient
- SSLClientServlet

Below, Example 1 shows a sample SSLClient, the relevant explanation is embeddede inside the code for easy understanding of the same.

Example 1: SSL Client Sample Code

```
package examples.security.sslclient;

import java.io.File;
import java.net.URL;
import java.io.IOException;
import java.io.InputStream;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.io.PrintStream;
import java.util.Hashtable;
import java.security.Provider;
import javax.naming.NamingException;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.servlet.ServletOutputStream;
import weblogic.net.http.*;
import weblogic.jndi.Environment;
/** SSLClient is a short example of how to use the SSL library of
 * WebLogic to make outgoing SSL connections. It shows both how to
 * do this from a stand-alone application as well as from within
 * WebLogic (in a Servlet).
 *
 */
```

```
public class SSLClient {
  public void SSLClient() {}
  public static void main (String [] argv)
    throws IOException {
    if ((!((argv.length == 4) || (argv.length == 5))) ||
       (!(argv[0].equals("wls")))
       ) {
      System.out.println("example: java SSLClient wls
              server2.weblogic.com 80 443 /examplesWebApp/SnoopServlet.jsp");
      System.exit(-1);
    }
    try {
      System.out.println("----");
      if (argv.length == 5) {
        if (argv[0].equals("wls"))
          wlsURLConnect(argv[1], argv[2], argv[3], argv[4], System.out);
      } else {  // for null query, default page returned...
        if (argv[0].equals("wls"))
          wlsURLConnect(argv[1], argv[2], argv[3], null, System.out);
      }
      System.out.println("----");
    } catch (Exception e) {
      e.printStackTrace();
      printSecurityProviders(System.out);
      System.out.println("----");
    }
  }
  private static void printOut(String outstr, OutputStream stream) {
    if (stream instanceof PrintStream) {
      ((PrintStream)stream).print(outstr);
      return;
    } else if (stream instanceof ServletOutputStream) {
      try {
        ((ServletOutputStream)stream).print(outstr);
        return;
      } catch (IOException ioe) {
        System.out.println(" IOException: "+ioe.getMessage());
      }
    }
    System.out.print(outstr);
  }
  private static void printSecurityProviders(OutputStream stream) {
    StringBuffer outstr = new StringBuffer();
    outstr.append(" JDK Protocol Handlers and Security Providers:\n");
    outstr.append("   java.protocol.handler.pkgs - ");
    outstr.append(System.getProperties().getProperty(
                                "java.protocol.handler.pkgs"));
    outstr.append("\n");
    Provider[] provs = java.security.Security.getProviders();
    for (int i=0; i<provs.length; i++)
       outstr.append("   provider[" + i + "] - " + provs[i].getName() +
                  " - " + provs[i].getInfo() + "\n");
    outstr.append("\n");
    printOut(outstr.toString(), stream);
  }
  private static void tryConnection(java.net.HttpURLConnection connection,
                                    OutputStream stream)
    throws IOException {
```

```
    connection.connect();
    String responseStr = "\t\t" +
                    connection.getResponseCode() + " -- " +
                    connection.getResponseMessage() + "\n\t\t" +
                    connection.getContent().getClass().getName() + "\n";
    connection.disconnect();
    printOut(responseStr, stream);
}
}
```

Using Two-Way SSL Authentication

When using certificate authentication, WebLogic Server sends a digital certificate to the requesting client. The client examines the digital certificate to ensure that it is authentic, has not expired, and matches the WebLogic Server instance that presented it.

With two-way SSL authentication (a form of mutual authentication), the requesting client also presents a digital certificate to WebLogic Server. When the instance of WebLogic Server is configured for two-way SSL authentication, requesting clients are required to present digital certificates from a specified set of certificate authorities. WebLogic Server accepts only digital certificates that are signed by root certificates from the specified trusted certificate authorities.

Q.9. Explain the benefit offered by EJB component architecture to application developers and customers in brief.

Ans**.: EJB:** Enterprise JavaBeans are software component models, their purpose is to build/support enterprise specific problems. EJB - is a reusable server-side software component. Enterprise JavaBeans facilitates the development of distributed Java applications, providing an object-oriented transactional environment for building distributed, multi-tier enterprise components. An EJB is a remote object, which needs the services of an EJB container in order to execute.

The primary goal of a EJB is WORA (Write Once Run Anywhere). Enterprise JavaBeans takes a high-level approach to building distributed systems.

EJB ARCHITECTURE: The Enterprise JavaBeans spec defines a server component model and specifies, how to create server-side, scalable, transactional, multiuser and secure enterprise-level components.

A typical EJB architecture consists of:

EJB clients: EJB client applications utilise the Java Naming and Directory Interface (JNDI) to look up references to home interfaces and use home and remote EJB interfaces to utilise all EJB-based functionality.

EJB home interfaces (and stubs): EJB home interfaces provide operations for clients to create, remove, and find handles to EJB remote interface objects.

EJB remote interfaces (and stubs): EJB remote interfaces provide business-specific client interface methods defined for a particular EJB.

EJB implementations: EJB implementations are the actual EJB application components implemented by developers to provide any application-specific business method invocation, creation, removal, finding, activation, passivation, database storage, and database loading logic.

Container EJB implementations (skeletons and delegates): The container manages the distributed communication skeletons used to marshal and unmarshal data sent to and from the client.

Benefit offered by EJB component architecture to application developers and customers:

Benefits to Application Developers:

The EJB component architecture is the backbone of the J2EE platform. The EJB architecture provides the following benefits to the application developer:

Simplicity: It is easier to develop an enterprise application with the EJB architecture than without it. Since, EJB architecture helps the application developer access and use enterprise services with minimal effort and time, writing an enterprise bean is almost as simple as writing a Java class. The application developer does not have to be concerned with system-level issues, such as security, transactions, multithreading, security protocols, distributed programming, connection resource pooling, and so forth. As a result, the application developer can concentrate on the business logic for domain-specific applications.

Application Portability: An EJB application can be deployed on any J2EEcompliant server. This means that the application developer can sell the application to any customers who use a J2EE-compliant server. This also means that enterprises are not locked into a particular application server vendor. Instead, they can choose the “best-of-breed” application server that meets their requirements.

Component Reusability: An EJB application consists of enterprise bean components. Each enterprise bean is a reusable building block. There are two essential ways to reuse an enterprise bean:

An enterprise bean not yet deployed can be reused at application development time by being included in several applications. The bean can be customised for each application without requiring changes, or even access, to its source code.

Other applications can reuse an enterprise bean that is already deployed in a customer's operational environment, by making calls to its client-view interfaces. Multiple applications can make calls to the deployed bean.

In addition, the business logic of enterprise beans can be reused through Java subclassing of the enterprise bean class.

Ability to Build Complex Applications: The EJB architecture simplifies building complex enterprise applications. These EJB applications are built by a team of developers and evolve over time. The component-based EJB architecture is well suited to the development and maintenance of complex enterprise applications. With its clear definition of roles and well-defined interfaces, the EJB architecture promotes and supports team-based development and lessens the demands on individual developers.

Separation of Business Logic from Presentation Logic: An enterprise bean typically encapsulates a business process or a business entity (an object representing enterprise business data), making it independent of the presentation logic. The business programmer need not worry about formatting the output; the Web page designer developing the Web page need be concerned only with the output data that will be passed to the Web page. In addition, this separation makes it possible to develop multiple presentation logic for the same business process or to change the presentation logic of a business process without needing to modify the code that implements the business process.

Easy Development of Web Services: The Web services features of the EJB architecture provide an easy way for Java developers to develop and access Web services. Java developers do not need to bother about the complex details of Web services description formats and XML-based wire protocols but instead can program at the familiar level of enterprise bean components, Java interfaces and data types. The tools provided by the container manage the mapping to the Web services standards.

Deployment in many Operating Environments— The goal of any software development company is to sell an application to many customers. Since, each customer has a unique

operational environment, he application typically needs to be customised at deployment time to each operational environment, including different database schemas.

The EJB architecture allows the bean developer to separate the common application business logic from the customisation logic performed at deployment.

The EJB architecture allows an entity bean to be bound to different database schemas. This persistence binding is done at deployment time. The application developer can write a code that is not limited to a single type of database management system (DBMS) or database schema.

The EJB architecture facilitates the deployment of an application by establishing deployment standards, such as those for data source lookup, other application dependencies, security configuration, and so forth. The standards enable the use of deployment tools. The standards and tools remove much of the possibility of miscommunication between the developer and the deployer.

Distributed Deployment: The EJB architecture makes it possible for applications to be deployed in a distributed manner across multiple servers on a network. The bean developer does not have to be aware of the deployment topology when developing enterprise beans but rather writes the same code whether the client of an enterprise bean is on the same machine or a different one.

Application Interoperability: The EJB architecture makes it easier to integrate applications from different vendors. The enterprise bean's client-view interface serves as a well-defined integration point between applications.

Integration with non-Java Systems: The related J2EE APIs, such as the J2EE Connector specification and the Java Message Service (JMS) specification, and J2EE Web services technologies, such as the Java API for XML-based RPC (JAXRPC), make it possible to integrate enterprise bean applications with various non- Java applications, such as ERP systems or mainframe applications, in a standard way.

Educational Resources and Development Tools: Since, the EJB architecture is an industry wide standard, the EJB application developer benefits from a growing body of educational resources on how to build EJB applications. More importantly, powerful application development tools

available from leading tool vendors simplify the development and maintenance of EJB applications.

Benefits to Customers

A customer's perspective on EJB architecture is different from that of the application developer. The EJB architecture provides the following benefits to the customer:

Choice of Server: Because the EJB architecture is an industry wide standard and is part of the J2EE platform, customer organisations have a wide choice of J2EEcompliant servers. Customers can select a product that meets their needs in terms of scalability, integration capabilities with other systems, security protocols, price, and so forth. Customers are not locked in to a specific vendor's product. Should their needs change, customers can easily redeploy an EJB application in a server from a different vendor.

Facilitation of Application Management: Because the EJB architecture provides a standardised environment, server vendors have the required motivation to develop application management tools to enhance their products. As a result, sophisticated application management tools provided with the EJB container allow the customer's IT department to perform such functions as starting and stopping the application, allocating system resources to the application, and monitoring security violations, among others.

Integration with a Customer's Existing Applications and Data: The EJB architecture and the other related J2EE APIs simplify and standardise the integration of EJB applications with any non-Java applications and systems at the customer operational environment. For example, a customer does not have to change an existing database schema to fit an application. Instead, an EJB application can be made to fit the existing database schema when it is deployed.

Integration with Enterprise Applications of Customers, Partners, and Suppliers: The interoperability and Web services features of the EJB architecture allows EJB-based applications to be exposed as Web services to other enterprises. This means that enterprises

have a single, consistent technology for developing intranet, Internet, and e-business applications.

Application Security: The EJB architecture shifts most of the responsibility for an application's security from the application developer to the server vendor, system administrator, and the deployer. The people performing these roles are more qualified than the application developer to secure the application. This leads to better security of operational applications.