


CMR Institute of Technology, Bangalore		
Department(s): Computer Science & Engineering		
Semester: 05	Section(s): A,B&C	
Subject: Advanced Java and J2EE		
Course Instructor : Ms.K.S. Shashikala/ Ms.Preethi Sheba		
Course duration: 29 July 2019 – 30 Nov 2019		
Date of Exam: 18 Nov 2019		
Advanced Java and J2EE IAT-3 Solution		

1. Define JSP. Explain different types of JSP tags (4 marks)

Answer:

Java Server Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application to set the cookies for the user preferences.

JSP tags:

A JSP code consists of a combination of HTML tags and JSP tags.

JSP tags define java code that is to be executed before the output of JSP program is sent to the browser.

A jsp tag begin with a <%, which is followed by java code , and ends with %>.

A JSP page has an HTML body with Java code embedded inside the JSP tags.

<%-- and --%> comment tag

<% and %> scriptlet tag

<%! and %> declaration tag

<%= and %> Expression tag

<%@ JSP directive tag

A simple JSP page which includes declarations tag, scriptlet tag, expressions tag, comments tag in it is shown here:

JSP CODE WITH ALL JSP TAGS

```

<html>
<head>
<title>JSP Example</title>
</head>
<body>
% @ page import="java.util.*"%>
<%-- This is a JSP example with scriptlets, comments , expressions --%>
<% out.println("This is a JSP Example"); %>

```

```

<% out.println("The number is "); %>
<%! int num12 = 12; int num32 = 12; %>
<%= num12*num32 %>
Today's date: <%= (new Date()).toLocaleString()%>
</body>
</html>

```

1b. Design a form to obtain two numbers from the user. Upon submission, write JSP code to add, subtract, multiply and divide the two numbers and display output to the user.

Answer:

index.html(form to get two numbers as input from the user)

```

<html>
  <head>
    <title>Enter two numbers </title>
  </head>

  <body>
    <form action="./add.jsp">
      First number: <input type="text" name="t1"/>
      Second number: <input type="text" name="t2"/>
      <input type="submit" value="SUBMIT" />
    </form>
  </body>
</html>

```

add.jsp

```

<html>
  <head>
    <title>Enter two numbers </title>
  </head>

  <body>
    <%= "<h1> The sum
is"+(Integer.parseInt(request.getParameter("t1"))+Integer.parseInt(request.getParameter("t2")
))+"</h1>"%>
    <%= "<h1> The difference is "+(Integer.parseInt(request.getParameter("t1"))-
Integer.parseInt(request.getParameter("t2")))+"</h1>"%>
    <%= "<h1> The product is
"+(Integer.parseInt(request.getParameter("t1"))*Integer.parseInt(request.getParameter("t2")))
+"</h1>"%>

```

```

<%= "<h1> The division result is
"+(Integer.parseInt(request.getParameter("t1"))/Integer.parseInt(request.getParameter("t2")))
+"</h1>"%>
    </body>
</html>

```

2 (a) What is a cookie? Which methods are required to create and retrieve a cookie in JSP?(3 marks)

Ans:

Cookie is a

- A small piece of information created by JSP program and stored on the client's hard disk by the browser.
- **response.addCookie()** and **request.getCookies()** will be used for adding the cookie to response header and getting the cookies in the request header respectively.
- Example to create and add a cookie is shown next.

userid.jsp – creates and adds a cookie

```

<HTML>
<HEAD>
    <TITLE> JSP Programming </TITLE>
</HEAD>
<BODY>
<%!
    String MyCookieName = "userID";
    String MyCookieValue = "JK1234";
    Cookie c=new Cookie(MyCookieName, MyCookieValue);
    %>
<%
response.addCookie(c);
    %>
</BODY>
</HTML>

```

readcookie.jsp –Example to read the cookies

```

<HTML>
<HEAD>
    <TITLE> JSP Programming </TITLE>
</HEAD>
<BODY>
    <%! String MyCookieName = "userID";
        String MyCookieValue;
        String CName, CValue;

```

```

        int found=0;
    %>
<%
    Cookie[ ] cookies = request.getCookies();
    for(int i=0; i<cookies.length; i++) {
        CName = cookies[i].getName();
        CValue = cookies[i].getValue();
        if(MyCookieName.equals(CName)) {
            found = 1;
            MyCookieValue = CValue;
        }
    }
    if (found ==1) { %>
        <P> Cookie name = <%= MyCookieName %> </P>
        <P> Cookie value = <%= MyCookieValue %> </P>
    <%}%>
</BODY>
</HTML>

```

b. Design a form to obtain user preference on a news page (national, sports, education, entertainment,...) using text field/drop down menu/check boxes. Store the preferred choice using a cookie called “newsCookie”.Retrieve and display the name and value of the cookie to the user

Answer:

How It Works ?

Our server sends some data to the visitor's browser in the form of a cookie. The browser may accept the cookie. If it does, it is stored as a plain text record on the visitor's hard drive. Now, when the visitor arrives at another page on your site, the browser sends the same cookie to the server for retrieval. Once retrieved, our server remembers what was stored earlier.

Cookies are a plain text data record of 5 variable-length fields –

- Expires
- Domain
- Path
- Name-Value

main.jsp

```

<%
    // Create cookies for the user preferences on the webpage
    Cookie pref1 = new Cookie("pref1", request.getParameter("pref1"));

```

```

Cookie pref2 = new Cookie("pref2", request.getParameter("pref2"));
Cookie pref3 = new Cookie("pref3", request.getParameter("pref3"));
// Set expiry date after 24 Hrs for both the cookies.
pref1.setMaxAge(60*60*24);
pref2.setMaxAge(60*60*24);
pref3.setMaxAge(60*60*24);
// Add both the cookies in the response header.
response.addCookie( pref1 );
response.addCookie( pref2 );
response.addCookie( pref3 );
%>
<html>
  <head>
    <title>Setting Cookies</title>
  </head>
  <body>
    <center>
      <h1>Setting Cookies</h1>
    </center>
    <ul>
      <li><p><b>Preference 1:</b>
        <%= request.getParameter("pref1")%>
      </p></li>
      <li><p><b>Preference 2:</b>
        <%= request.getParameter("pref2")%>
      </p></li>
      <li><p><b>Preference 3:</b>
        <%= request.getParameter("pref3")%>
      </p></li>
    </ul>
  </body>
</html>
Cookie cookie = null;
Cookie[] cookies = null;
// Get an array of Cookies associated with the this domain

```

```

cookies = request.getCookies();
if( cookies != null ) {
out.println("<h2> Found Cookies Name and Value</h2>");
for (int i = 0; i < cookies.length; i++) {
cookie = cookies[i];
out.print("Name : " + cookie.getName( ) + ", ");
out.print("Value: " + cookie.getValue( )+" <br/>");
}
} else {
out.println("<h2>No cookies founds</h2>");
}
%>
</body>
</html>

```

hello.jsp

```

<html>
<body>
<form action = "main.jsp" method = "GET">
Enter news preference 1: <input type = "text" name = "pref1">
<br />
Enter news preference 2: <input type = "text" name = "pref2" />
Enter news preference 3: <input type = "text" name = "pref3" />
<input type = "submit" value = "Submit" />
</form>
</body>
</html>

```

3 (a) What is HttpSession? Discuss other common techniques for tracking session.(3 marks)

The HttpSession interface enables a servlet to read and write the state information that is associated with an HTTP session.

Several of its methods are summarized in Table .

All of these methods throw an IllegalStateException if the session has already been invalidated.

- long [getCreationTime\(\)](#)
Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
- String [getId\(\)](#)
Returns a string containing the unique identifier assigned to this session.

long [getLastAccessedTime\(\)](#)

The commonly used methods to track sessions as a client moves between HTML pages and JSP programs are:

1. By using a hidden field
2. By using a cookie
3. By using a JavaBean

Hidden Form Fields

A web server can send a hidden HTML form field along with a unique session ID as follows :

```
<input type = "hidden" userid = "sessionid" value = "12345">
```

This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or the POST data.

Each time the web browser sends the request back, the session_id value can be used to keep the track of different web browsers.

3b. Create a login form. Check whether the username and password is equal to “admin” and “a123” respectively. If login is successful, create a session and set a session attribute called “username” to “admin”.

Display a customized welcome message to “admin” by retrieving the session attribute. (7 marks)

Answer:

index.jsp

For every user there will be a particular session, here we are validating the details of a user and setting the user in a session.

```
<html>
  <head>
    <title>Session In JSP</title>
  </head>
  <body>
    <h1>Session in JSP</h1>
    <h2>USER LOGIN SESSION</h2>
```

```
<form action="validate.jsp" method="post">
```

```
<!-- Here we are taking the values from user and triggering the validate.jsp file -->
```

```

<table>
<tr><td>USER NAME</td><td><input type="text" name="username"></td></tr>
<tr><td>PASSWORD</td><td><input type="password" name="password"></td></tr>
<tr><td></td></tr>
<td><button type="submit">LOGIN</button></td></tr>
</table>

```

```

</form>
</fieldset>
</body>
</html>

```

Validate.jsp

Here we are validating the particular user by getting the input values, and sending them to its respective page.

```

<html>
<head>
<title>Validate</title>
</head>
<body>
<!-- values given at login page are taken here and checks if the valid details are not -->
<%
String username=request.getParameter("username");
String password=request.getParameter("password");
if(username.equals("admin") && password.equals("a123")){
//if the user is valid, then this block executes and WE ARE KEEPING THE USER IN A
SESSION
session.setAttribute("user", username);
// WE DECLARE THE USER IN A SESSION
response.sendRedirect("logged.jsp"); //AND WE REDIRECT TO LOGIN PAGE
}
else{

//IF THE USER IS NOT AUTHORISED THEN AGAIN HE WILL BE REDIRECTED TO
THE SAME LOGIN PAGE
response.sendRedirect("index.jsp");

```



```
}
%>
</body>
</html>
```

Logged.jsp

Soon as the validation done, if the user is authorized according to our condition will be redirected to login page else forwarded to the same login page.

```
<html>
<head>
<title>Success</title>
</head>
<body>
<%
//HERE WE GETTING THE ATTRIBUTE DECLARED IN VALIDATE.JSP AND
CHECKING IF IT IS NULL, THE USER WILL BE REDIRECTED TO LOGIN PAGE
String uid = (String)session.getAttribute("user");
if (uid == null)
{
%>
<!-- NOT A VALID USER, IF THE USER TRIES TO EXECUTE LOGGED IN PAGE
DIRECTLY, ACCESS IS RESTRICTED -->
<jsp:forward page="index.jsp"/>
<%
}
else
{
//IF THE VALUE IN SESSION IS NOT NULL THEN THE IS USER IS VALID
out.println(" <h1>WELCOME ADMIN "</h1>");%>
<% }
%>
</body>
</html>
```

4 a.Explain the three methods in JSP that are automatically called.

Answer:

The lifecycle of JSP is controlled by three methods which are automatically called when a JSP is requested and when the JSP terminates normally.

These are:

jspInit () , jspService() , jspDestroy().

jspInit() method is identical to the init() method in a Java Servlet and in applet.

It is called first when the JSP is requested and is used to initialize objects and variables that are used throughout the life of the JSP.

_jspService() method is automatically called and retrieves a connection to HTTP.

It will call doGet or doPost() method of servlet created.

jspDestroy() method is identical to the destroy() method in Servlet.

The destroy() method is automatically called when the JSP terminates normally.

4b. Write a JSP program to print the numbers, 1 to 10, its square and its cube in separate columns in a tabular format.

Ans:

```
<html>
<title>No</title>
<body>
<h1>No and its square/cube</h1>
<table>
<%
out.println("<td><tr>No </tr><tr>square </tr><tr>cube</tr></td><br />");
for ( int i =1; i<=10 ; i++)
{
%>
<tr>
<td> <%=i%></td>
<td><%=i*i%> </td>
<td> <%=i*i*i%></td>
</tr>
<%
}
%>
</table>
</body>
</html>
```

5a. List and explain various JDBC driver types (6 marks)

Ans:

- JDBC driver specification classifies JDBC drivers **into four groups**.
- Each group is referred to as a JDBC driver type and addresses a specific need for communicating with various DBMSs.
- Type 1 JDBC – to – ODBC Driver
 - Also known as JDBC/ODBC Bridge used to translate DBMS calls between the JDBC specification.
 - Those messages are translated by the JDBC – to – ODBC driver into the ODBC message format, which is then translated into the message format understood by the DBMS.
- Type 2 Java/Native Code Driver
 - Uses Java classes to generate platform specific code – that means, code only understood by the specific DBMS.
- Type 3 JDBC Driver
 - Also known as Java Protocol, which converts SQL queries into JDBC formatted statements.
 - The JDBC formatted statements are translated into the format required by the DBMS.
- Type 4 JDBC Driver
 - Also known as Type 4 database protocol, which converts SQL queries into the format required by the DBMS.

**5b. Assume there exists a table called “books” with fields isbn(String), title(String), author(String), publisher(String), price(Real)
Assume the Connection object “conn” holds a connection to the database in which the table is present. Write code snippet to display all records in the table “books” in tabular format.(4 marks)**

Answer:

```
mysql> use test;
```

```
Database changed
```

```
mysql> create table books(ISBN VARCHAR(30),TITLE VARCHAR(30),AUTHOR  
VARCHAR(30),PUBLISHER VARCHAR(30),PRICE REAL);
```

```
mysql> insert into books values('Book1','Taming the dragon','chetan bhagat','tata mcgraw  
hill',300.50);
```

CODE:

```
import java.sql.*;
```

```
public class Odbc {
```

```
public static void main(String[] args)throws ClassNotFoundException,SQLException {
```

```

Class.forName("com.mysql.jdbc.Driver");
Connection con = null;
con=DriverManager.getConnection("jdbc:mysql://localhost/test","root","shashi");
PreparedStatement pstmt;
ResultSet rs; String title;
pstmt = con. prepareStatement("SELECT Price, ISBN, TITLE, AUTHOR, PUBLISHER
FROM BOOKS WHERE PRICE<?");
pstmt.setInt(1,500);
// Create a PreparedStatement object
rs = pstmt.executeQuery(); // Get the result table from the query
while (rs.next()) { // Position the cursor
title = rs.getString(3); // Retrieve the Second column value
System.out.println("Book title= " + title );
// Print the column values
}
rs.close(); // Close the ResultSet
pstmt.close(); // Close the PreparedStatement
}
}

```

6a.Explain the basic steps involved in a database connection with code snippets. Assume you are connecting to a database named customerDB, username =root and password="root123".

Answer:

The process used by J2EE components for interacting with a DBMS is divided into the following steps:

1. Loading the JDBC driver

- The jdbc driver must be loaded before the J2EE component can be connected to the database.

The Class.forName() method is used to load the JDBC driver.

- Driver is loaded by calling the method and passing it the name of driver .

Eg. If a developer's J2EE component interacts with Microsoft Access,the routine must load the JDBC/ODBC Bridge driver.

Eg. for loading ODBC Driver,
Class. forName("sun;jdbc.odbc.JdbcOdbcDriver");

For loading mysql driver,
Class.forName("com.mysql.jdbc.Driver");

2. Connecting to the DBMS.

- Once the driver is loaded , J2EE component must connect to the DBMS using DriverManager.getConnection() method.

- The java.sql.DriverManager class is the highest class in hierarchy and is responsible for managing driver information.

getConnection() method, takes three arguments ,**the URL of the database, User, Password of the DBMS.**

- It returns a **Connection interface object** that is used through out the process to reference a database .

The java.sql.Connection interface object sends the statements to the DBMS for processing.

```
Class.forName("com.mysql.jdbc.Driver");
```

```
Connection con=
```

```
DriverManager.getConnection("jdbc:mysql://localhost/user","root","tiger");
```

Here user is database name,

root is username of DBMS

tiger is password of DBMS

Once a connection is obtained, we can interact with the database.

The **JDBC Statement, CallableStatement, and PreparedStatement interfaces** define the methods and properties that help to send SQL or PL/SQL commands and receive data from our database.

They also define methods that help bridge data type differences between Java and SQL data types used in a database.

Summary of each interface's purpose to decide on the interface to use.

Statement Use this for general-purpose access to your database.

Useful when you are using static SQL statements at runtime. The Statement interface cannot accept parameters.

PreparedStatement Use this when you plan to use the SQL statements many times. The

PreparedStatement interface accepts input parameters at runtime.

CallableStatement Use this when you want to access the database stored procedures. The

CallableStatement interface can also accept runtime input parameters

Step 3:

Creating and Executing a statement.

- The next step after the JDBC is loaded and connection is successfully made with a particular database managed by the DBMS is to send a SQL query to the DBMS for processing.

- SQL query consists series of SQL commands that direct DBMS to do something ,For Example Return rows of data to the J2EE component.

A Statement is an interface that represents an SQL statement.

The Connection object's **createStatement() method is used to create a Statement object to execute a SQL statement.**

- **Connection.createStatement() method** is used to create a Statement Object.
- The Statement object is then used to execute a query and return a ResultSet object that contains the response from the DBMS .

6b. What is a connection pool? Discuss advantages of a connection pool. (3 marks)

Ans:

In a connection pool, connecting to a database is performed on a per – client basis.

That is, each client must open its own connection to a database and the connection cannot be shared with unrelated clients.

- For example, a client that needs to frequently interact with a database must either open a connection and leave the connection open during processing, or open or close and reconnect each time the client needs to access the database.
- Leaving a connection open might prevent another client from accessing the database should the DBMS have available a limited number of connections.
- Connecting and reconnecting a simply time – consuming and causes performance degradation.

7a. List and explain various statement objects in JDBC

Once a connection to the database is opened, the J2EE component creates and sends a query to access data contained in the database.

- There are three Statement objects:
 - Statement
 - Executes a query immediately.
 - PreparedStatement
 - Used to execute a compiled query.
 - CallableStatement
 - Used to execute store procedures.
- The Statement object is used whenever a J2EE component needs to immediately execute a query without first having the query compiled.
- The Statement object contains the executeQuery() method, which is passed the query as an argument.
- The query is then transmitted to the DBMS for processing.
- The **executeQuery()** method returns one ResultSet object that contains rows, columns, and metadata that represent data requested by query.
- The ResultSet object also contains methods that are used to manipulate data in the ResultSet.
- The **execute() method** of the Statement object is used when there may be multiple results returned.
- A third commonly used method of the Statement object is the executeUpdate() method.
- The **executeUpdate() method** is used to execute queries that contain UPDATE and DELETE SQL statements, which changes values in a row and removes a row respectively.
- The executeUpdate() method returns an integer indicating the number of rows that were updated by the query.
- The executeUpdate() is used to INSERT, UPDATE, DELETE statements.
- **A SQL query can be precompiled and executed by using the PreparedStatement object.**
- **The query is constructed similar to the queries in previous object.**

- **A question mark is used as a placeholder for a value that is inserted into the query after the query is compiled.**
- The callable Statement object is used to call a stored procedure from within a J2EE object. A stored procedure is a block of code and is identified by a unique name. The code can be written in Transact-C, PL/SQL.
- Stored procedure is executed by invoking by the name of procedure.
- **The callableStatement uses three types of parameter when calling stored procedure. The parameters are IN, OUT, INOUT**

7b. Assume there exists a table called "books" with fields isbn(String), title(String), author(String), publisher(String), price(Real). Assume the Connection object "conn" holds a connection to the database in which the table is present.

Write a parameterized query to retrieve and display all books whose price is below Rs. 500 (Use PreparedStatement)

Ans:

```
import java.sql.*;

public class Odbc {

public static void main(String[] args) throws ClassNotFoundException, SQLException {

    Class.forName("com.mysql.jdbc.Driver");

    Connection con = null;

    con=DriverManager.getConnection("jdbc:mysql://localhost/test","root","shashi");

    PreparedStatement pstmt;

    ResultSet rs; String title;

pstmt = con. prepareStatement("SELECT Price, ISBN, TITLE, AUTHOR, PUBLISHER
FROM BOOKS WHERE PRICE<?");

    pstmt.setInt(1,500);

    // Create a PreparedStatement object

rs = pstmt.executeQuery();    // Get the result table from the query

while (rs.next()) {          // Position the cursor

    title = rs.getString(3);    // Retrieve the Second column value

    System.out.println("Book title= " + title );

    // Print the column values

}

rs.close();                // Close the ResultSet

pstmt.close();            // Close the PreparedStatement

}

}
```

8a. What is a transaction? Explain the various methods used to control transaction in JDBC. [05 marks]

Ans:

- A database transaction consists of a set of SQL statements, each of which must be successfully completed for the transaction to be completed.
- If one fails, SQL statements that executed successfully up to that point in the transaction must be rolled back.
- A database transaction isn't completed until the J2EE component calls the **commit() method of the Connection object.**

AutoCommit and rollback()

- DBMS has an AutoCommit feature that is set to true by default. **The commit() method is automatically called due to this feature.**
- If a J2EE component is processing a transaction, the AutoCommit feature must be deactivated by calling the **SetAutoCommit method and passing it a false parameter.**
- If any issue occurs after the commit you can revert all the changes done till this commit by invoking the **rollback()** method.

Eg:

```
Con.rollback()
```

where Con is the Connection object.

```
import java.sql.*;
public class Jdbc1
{
public static void main(String[] args) throws ClassNotFoundException, SQLException {

Class.forName("com.mysql.jdbc.Driver");
Connection con=
DriverManager.getConnection("jdbc:mysql://localhost/CustomerInfo","root","shashi");
con.setAutoCommit(false);
Statement stmt=con.createStatement();
stmt.executeUpdate("insert into customers values(90,'Shradha','sharma')");
stmt.executeUpdate("insert into customers values(91,'Shilpa','sharadh')");

con.commit();
con.close();

}
}
```

8b. Write a program to demonstrate transaction processing for the following problem.

Assume there exists a table called "accounts" with fields accountNumber(String), balance(Real).

Assume the Connection object "conn" holds a connection to the database in which the table is present.

The transaction consists of two queries

1. Debit Rs.2000 from accountNumber with value 101

Credit Rs. 2000 to accountNumber with value 102.

Ans:


```

import java.sql.*;
public class Jdbc1
{
public static void main(String[] args) throws ClassNotFoundException,SQLException {
Class.forName("com.mysql.jdbc.Driver");
Connection con = null;
try {
    // 1st Step, Make a connection
    con=DriverManager.getConnection("jdbc:mysql://localhost/CustomerInfo","root","s
hashi");
}
catch (SQLException e) {
    System.err.println("There was an error getting the connection");
}
try {
    // 2nd Step, Disable the auto commit
    con.setAutoCommit(false);
    System.err.println("The autocommit was disabled!");
} catch (SQLException e) {
    System.err.println("There was an error disabling autocommit");
}

// Starts JDBC Transaction
try {
    // 3rd Step, Execute the statements
    Statement stmt=con.createStatement();
    stmt.executeUpdate("UPDATE accounts set bal = bal-2000 WHERE accountno = 101;");
    stmt.executeUpdate("UPDATE accounts set bal = bal+2000 WHERE accountno =
102;");
    // 4th Step, Complete a transaction, committing the changes.
    con.commit();
    System.err.println("The transaction was successfully executed");
} catch (SQLException e) {
    try {
// Final Step, We must rollback the transaction if a SQLException occurs
        con.rollback();
        System.err.println(e.getMessage());
        System.err.println("Transaction rollback");
    }
    catch (SQLException e1) {
        System.err.println(e1.getMessage());
        System.err.println("There was an error making a rollback");
    }
}
}
}
}

```