CBCS SCHEME

USN | 1 | C | R | 2 | 4 | M | C | 0 | 1 | 0 |

MMCB311F

Third Semester MCA Degree Examination, Dec.2025/Jan.2026
Web Programming using Java

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

			M	L	C
		Module – 1			
Q.1	a.	Explain HTTP request headers.	10	L1	CO1
	b.	Describe DOM structure in Javascript.	10	L2	CO1
		OR			
Q.2	a.	Explain the architecture of web applications?	10	L1	CO1
	b.	Develop an HTML program illustrating the use of different CSS selectors.	10	L2	CO1
		Module – 2			
Q.3	a.	Describe the lifecycle methods of a Servlet with syntax.	10	L2	CO2
	b.	What is a Cookie? Explain the creation of Cookie and retrieving information from Cookie with code snippets.	10	L2	CO2
		OR			
Q.4	a.	Differentiate the ways of redirecting requests using sendRedirect() and forward()	10	L2	CO2
	b.	Write a Servlet program to read data from a HTML form (branch data from radio buttons and subjects data using checkbox) and display.	10	L3	CO2
		Module – 3			
Q.5	a.	What are different types of tags in JSP demonstrate with an example.	10	L2	CO3
	b.	What is JSP? Explain the advantages of JSP over Servlet.	10	L2	CO3
		OR			
Q.6	a.	What are the different implicit objects in JSP? Explain with an example.	10	L2	CO3
	b.	Write a jsp program to demonstrate the attributes of page directive.	10	L3	CO3
		Module – 4			
Q.7	a.	List and explain different JDBC Driver types along with its advantages and dis advantages?	10	L2	CO4
	b.	Illustrate the steps involved in connecting a java program to a database with code snippets.	10	L2	CO4
		OR			
Q.8	a.	Discuss how the methods of the `Statement` interface are used in JDBC, with an appropriate example.	10	L2	CO4
	b.	Create a Java JDBC program that demonstrates CRUD operations on a `Book` database table containing `book_id`, `book_name`, and `book_author`.	10	L3	CO4
		Module – 5			
Q.9	a.	Describe the architecture of MVC?	10	L2	CO5
	b.	Explain the concept of View in the MVC architecture. Discuss its roles, responsibilities, and need in web applications.	10	L2	CO5
		OR			
Q.10	a.	Explain the request–response flow handled by a Controller in Spring MVC.	10	L2	CO5
	b.	Discuss the role of model in MVC application	10	L2	CO5

* * * * *

1.a. Explain HTTP request headers.

1. Host

- **Specifies the domain name of the server.**

- **Mandatory in HTTP/1.1.**
  Host: www.example.com

2. User-Agent

- **Identifies the client application, browser, and operating system.**
  User-Agent: Mozilla/5.0

3. Accept

- **Tells the server which media types the client can accept.**
  Accept: text/html, application/json

4. Accept-Language

- **Indicates the preferred language for the response.**
  Accept-Language: en-US

5. Accept-Encoding

- **Specifies acceptable compression methods.**

Accept-Encoding: gzip, deflate

6. Content-Type

- **Describes the format of data sent in the request body.**
- **Content-Type: application/json**

7. Content-Length

- **Indicates the size of the request body in bytes.**
  Content-Length: 256

8. Authorization

- **Contains credentials for user authentication.**
  Authorization: Bearer <token>

9. Cookie

- **Sends stored cookies from client to server.**
  Cookie: sessionId=12345

10. Referer

- **Specifies the URL of the previous page.**

Referer: https://example.com/home

11. Connection

Controls whether the connection remains open.
Connection: keep-alive

1.b Describe DOM structure in Javascript.

The DOM is an application programming interface (API) that defines an interface between XHTML documents and application programs.

- It is an abstract model because it must apply to a variety of application programming languages.
- Each language that interfaces with the DOM must define a binding to that interface.
- The actual DOM specification consists of a collection of interfaces, including one for each document tree node type.
- They define the objects, methods, and properties that are associated with their respective node types.
- With the DOM, users can write code in programming languages to create documents, move around in their structures, and change, add, or delete elements and their content.
- Documents in the DOM have a treelike structure, but there can be more than one tree in a document.

2.a. Explain the architecture of Java Web Applications.

Presentation Layer (Client Layer)

- Responsible for user interface and interaction
- Sends HTTP requests to the server
- Receives and displays responses

Technologies:

- HTML, CSS, JavaScript, React, Angular
- Application / Business Logic Layer
- Processes client requests

Application Layer:

- Implements business rules and validations
- Coordinates between presentation and data layers

Technologies:
Java Servlets, JSP, Spring MVC, Spring Boot, Node.js
Data Layer (Database Layer)

- Stores and manages application data
- Performs CRUD operations

Technologies:
MySQL, Oracle, PostgreSQL, MongoDB

2.b. Develop an HTML program illustrating the use of different CSS selectors.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>CSS Selectors Example</title>
   <style>
      /* 1. Universal Selector */
      * {
         font-family: Arial, sans-serif;
      }
      /* 2. Element Selector */
      p {
         color: blue;
      }
      /* 3. ID Selector */
      #mainHeading {
         color: darkred;
         text-align: center;
      }
      /* 4. Class Selector */
      .highlight {
         background-color: yellow;
         font-weight: bold;
      }
      /* 5. Group Selector */
      h2, h3 {
         color: green;
      }
      /* 6. Descendant Selector */
      div p {
         color: purple;
      }
      /* 7. Child Selector */
      ul > li {
         color: brown;
      }
      /* 8. Attribute Selector */
      input[type="text"] {
```

```
        border: 2px solid blue;
    }
    /* 9. Pseudo-class Selector */
    a:hover {
        color: red;
    }
    /* 10. Pseudo-element Selector */
    p::first-letter {
        font-size: 24px;
        color: red;
    }
  </style>
</head>
<body>
  <h1 id="mainHeading">CSS Selectors Demonstration</h1>
  <h2>Basic Selectors</h2>
  <p>This paragraph uses the element selector.</p>
  <p class="highlight">This paragraph uses the class selector.</p>
  <h3>Descendant Selector Example</h3>
  <div>
    <p>This paragraph is inside a div.</p>
  </div>
 <h3>Child Selector Example</h3>
  <ul>
    <li>Item One</li>
    <li>Item Two</li>
  </ul>
  <h3>Attribute Selector Example</h3>
  <input type="text" placeholder="Enter your name">
  <h3>Pseudo-class Selector</h3>
  <a href="#">Hover over this link</a>
</body>
</html>
```

3.a. Describe the lifecycle methods of a Servlet with syntax.

A. Servlets are programs that run on a Web or Application server act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

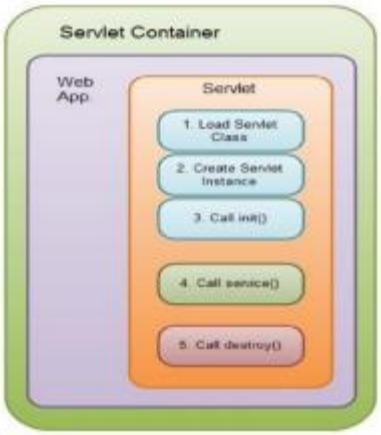
Servlets are server side components that provide a powerful mechanism for developing web applications.

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet

The servlet is initialized by calling the init () method.

The servlet calls service() method to process a client's request.

The servlet is terminated by calling the destroy() method.
Finally, servlet is garbage collected by the garbage collector of the JVM.

5. 

The init() method :

- The init method is designed to be called only once.
- It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.
- The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this:

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

The service() method :

- The service() method is the main method to perform the actual task.
- The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service.
- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Signature of service method:

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException
```

{
}

The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc.methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.
The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
// Servlet code
}
```

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet.
This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.
After the destroy() method is called, the servlet object is marked for garbage collection.
The destroy method definition looks like this:

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

3.b. What is a Cookie? Explain the creation of Cookie and retrieving the information from Cookie with code snippets.

Cookies are small bits of textual information that a web server sends to a browser and that the browser later returns unchanged when visiting the same web site or domain

Sending cookies to the client:

1.Creating a cookie object: Create an object for the cookie class

• Cookie():constructs a cookie.
• Cookie(String name, String value)constructs a cookie with a specified name and value.

EX:

```
Cookie ck=new Cookie("user",”mca");
```

2.Setting the maximum age

setMaxAge() is used to specify how long (in seconds) the cookie should be valid.

Ex:cookie.setMaxAge(60*60*24);

3.Placing the cookie into the HTTP response headers.

We use response.addCookie to add cookies in the HTTP response header as follows:

response.addCookie(cookie);

Reading cookies from the client:

1. Call request.getCookies(). This yields an array of cookie objects.
2. Loop down the array, calling getName on each one until you find the cookie of interest.

Ex:

```
String cookieName="userID";
Cookie[] cookies=request.getCookies();
If(cookies!=null)
{
for(int i=0;i<cookies.length;i++){
Cookie cookie=cookies[i];
if(cookieName.equals(cookie.getName())){
doSomethingwith(cookie.getValue());
}}}
```

4.a. Differentiate the ways of redirecting requests using sendRedirect() and forward()

Feature	`sendRedirect()`	`forward()`
Type of redirection	Client-side	Server-side
Method used	`response.sendRedirect()`	`RequestDispatcher.forward()`
Number of requests	Two requests	One request
URL change in browser	Yes	No
Performance	Slower	Faster
Can access request attributes	No	Yes
Can redirect to external URL	Yes	No
Data sharing	Not possible	Possible
Typical use	After form submission	Internal page navigation

Example:	protected void doGet(HttpServletRequest request, HttpServletResponse response)
	throws IOException {
	response.sendRedirect("success.html");
	}
		protected void doGet(HttpServletRequest request,
		HttpServletResponse response)
		throws ServletException, IOException {
		request.setAttribute("msg", "Welcome User");
		RequestDispatcher rd =
		request.getRequestDispatcher("welcome.jsp");
		rd.forward(request, response);
		}

4.b. Write a servlet program to read data from a HTML form(branch data from radio buttons and subjects data using checkbox) and display.
Servlet code

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String branch = request.getParameter("branch");
String[] subjects = request.getParameterValues("subject");
out.println("<html><body>");
out.println("<h2>Submitted Information</h2>");
```

```
if (branch != null)
out.println("<p><b>Branch:</b> " + branch+ "</p>");
else
out.println("<p><b>Branch:</b> Not Selected</p>");
if (subjects != null) {
out.println("<p><b>Selected Subjects:</b></p>");
out.println("<ul>");
for (String subject : subjects) {
out.println("<li>" + subject + "</li>");
}
out.println("</ul>");
} else {
out.println("<p><b>Selected Subjects:</b> None</p>");
}
out.println("</body></html>");
out.close();
}
```

HTML Form

```
<body>
<h2>Choose Your Gender and Favorite Colors</h2>
<form action="DisplayDataServlet" method="post">

<label><b>Branch:</b></label><br>
<input type="radio" name="gender" value="MCA"> MCA<br>
<input type="radio" name="gender" value="MBA"> MBA<br><br>
<label><b>Subjects:</b></label><br>
<input type="checkbox" name="subject" value="C"> C<br>
<input type="checkbox" name="subject" value="Java"> Java<br>
<input type="checkbox" name="subject" value="Python"> Python<br>
<input type="checkbox" name="subject" value="R"> R<br><br>
<input type="submit" value="Submit"></form></body>
```

5.a. what are different types of tags in JSP Demonstrate with an example.

JSP scriptlet tag:A scriptlet tag java source code in JSP.

<% java source code %>

JSP Declaration Tag: The JSP declaration tag is used to declare variables, objects and methods. The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

<%! Field or method declaration %>

JSP Expression Tag:

Expression Tag is used to print out java language expression that is put between the tags. An expression tag can hold any java language expression that can be used as an argument to the out.print() method.

Syntax :<%= java expression %>

JSP Directives:

The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.

Syntax

<%@ directive attribute=”value” %>

5.b. What is JSP? Explain the advantages of JSP over Servlet?

• It has all the features of servlet and additionally has implicit objects, predefined tags,
custom tags
• It is easily managed. It separates business logic with presentation logic
• It can be easily deployed
• If JSP page is modified, no need to redeploy but if changes are required in servlet , the entire code needs to be updated and recompile

JSP	Servlet
JSP is a web page scripting language that can generate dynamic content	Servlets are Java programs that already compiled which also creates dynamic web content
JSP runs slower compared to servlet as it takes compilation time to convert into servlets	Servlets run faster compared to JSP.
It's easier to code in JSP than in Java Servlets.	It is not easier when compared to JSP
In MVC, jsp act as a view.	In MVC, servlet act as a controller.
JSP are generally preferred when there is not much processing of data required.	servlets are best for use when there is more processing and manipulation involved.
The advantage of JSP programming over servlets is that we can build custom tags which can directly call Java beans.	There is no such custom tag facility in servlets.
We can achieve functionality of JSP at client side by running JavaScript at client side.	There are no such methods for servlets.

6.a. What are the different implicit objects in JSP? Explain with an example.

JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out.

It is the object of JspWriter.

- In case of servlet you need to write:

  PrintWriter out=response.getWriter();

- But in JSP, you don't need to write this code.

E**xample of out implicit object**

- <html>
- <body>
- <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
- </body>
- </html>

Request :

- The JSP request is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container.
- It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.
- It can also be used to set, get and remove attributes from the jsp request scope.

index.html

- <form action="welcome.jsp">
- <input type="text" name="uname">
- <input type="submit" value="go">
- </form>
- welcome.jsp
- <%
- String name=request.getParameter("uname");
- out.print("welcome "+name); %>

Response Object :

- In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.

index.html

- **<form** action="welcome.jsp">
- **<input** type="text" name="uname">
- **<input** type="submit" value="go"><**br/>**
- **</form>**
- **welcome.jsp**
- <%
- response.sendRedirect("http://www.google.com");
- %>

Config:

- In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.
- Generally, it is used to get initialization parameter from the web.xml file.
- **index.html**

```
<form action="welcome">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>
```

- **web.xml file**

```
<web-app>

<servlet>

<servlet-name>sonoojaiswal</servlet-name>

<jsp-file>/welcome.jsp</jsp-file>

<init-param>

<param-name>dname</param-name>

<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>sonoojaiswal</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>
```

welcome.jsp

- <%
- out.print("Welcome "+request.getParameter("uname"));
-
- String driver=config.getInitParameter("dname");
- out.print("driver name is="+driver);
- %>

Application:

- In JSP, application is an implicit object of type *ServletContext*.
- The instance of ServletContext is created only once by the web container when application or project is deployed on the server.
- This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

This initialization parameter can be used by all jsp pages.

Session:

- In JSP, session is an implicit object of type HttpSession. The Java developer can use this object to set,get or remove attribute or to get session information.
- **index.html**
- <html>
- <body>
- <form action="welcome.jsp">
- <input type="text" name="uname">
- <input type="submit" value="go">
- </form>
- </body>

</html>

Pagecontext:

- In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:page
- request
- session
- application
- In JSP, page scope is the default scope.
- **index.html**
- <html>

- <body>
- <form action="welcome.jsp">
- <input type="text" name="uname">
- <input type="submit" value="go">
- </form>
- </body>
- </html>

Page:

- In JSP, page is an implicit object of type Object class.
- This object is assigned to the reference of auto generated servlet class.
- It is written as:Object page=this;
- For using this object it must be cast to Servlet type.
- For example:<% (HttpServlet)page.log("message"); %>
- Since, it is of type Object it is less used because you can use this object directly in jsp.For example:<% this.log("message"); %>

Exception:

- In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.It is better to learn it after page directive. Let's see a simple example:
- **error.jsp**
- <%@ page isErrorPage="true" %>
- <html>
- <body>
-
- Sorry following exception occured:<%= exception %>
-
- </body>
- </html>

6.b. Write a JSP program to demonstrate the attributes of Page directive.

student.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Student Information System</title>
<h4>Enter the details</h4>
```

```
</head>
<body>
<form action="process.jsp" method="post">
<table boder=1>
<tr><td>Usn No.</td><td><input type="text" name="usn"/></td></tr>
<tr><td>Student Name</td><td><input type="text" name="name"/></td></tr>
<tr><td>Department</td><td><input type="text" name="dept"/></td></tr>
</table>
<input type="submit" value="Submit"/>
<input type="reset" value="Clear"/>
</form>
</body>
</center>
</html>
```

process.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<% String
name="",usn="",dept=
"";
usn=request.getParame
ter("usn");
name=request.getPara
meter("name");
dept=request.getParam
eter("dept");
out.println("<html><center><body bgcolor=grey>"); %>
<%@page errorPage="error.jsp" session="true" isThreadSafe="true" %>
<%synchronized(this)
{
wait(1000);
}
if(dept.equals("")||name.equals("")||usn.equals(""))
{
throw new RuntimeException("FieldBlank");
```

```
}else
   {
   session.setAttribute("name",name);
   session.setAttribute("usn",usn);
   session.setAttribute("dept",dept);
   request.getRequestDispatcher("display.jsp").forward(request,response);
   }
    %>
   <%out.println("<body></center></html>");
    %>
   </body>
   </html>
```

error.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%@page isErrorPage="true"%>
<%=exception %>
</body>
</html>
```

display.jsp

```
<%@page import="java.util.*" session="true" contentType="text/html;"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
<h3 align="center"> Student information</h3>
<h4 align="right"><%= new Date() %></h4>
</head>
<center>
<body>
<table border=1 cellPadding=10 cellSpacing=10>
```

```
<tr>
<th>Name</th>
<th>USN</th>
<th>Dept</th>
</tr>
<tr>
<td><%=session.getAttribute("usn")%></td>
<td><%=session.getAttribute("name")%></td>
<td><%=session.getAttribute("dept")%></td>
</tr>
</table>
</body>
<a href="student.jsp"> Back to info</a>
</center>
</html>
```

7.a. List and explain different JDBC driver types along with its advantages and disadvantages.

Type 1 Driver – JDBC–ODBC Bridge Driver

Description

Converts **JDBC method calls into ODBC function calls**

Provided by Sun as sun.jdbc.odbc.JdbcOdbcDriver

Uses **ODBC drivers** installed on the client machine

Architecture (Flow)

Java Application → JDBC API → JDBC-ODBC Bridge → ODBC Driver → Database

Advantages

Can connect to almost any database that supports ODBC

Simple and easy to use

Disadvantages

Platform dependent

Requires ODBC installation on client machine

Not suitable for web applications or applets

Slower performance

Type 2 Driver – Native API Driver (Partly Java Driver)

Description

Converts JDBC calls into **database-specific native calls**

Uses vendor-provided client libraries

Architecture (Flow)

Java Application → JDBC API → Type 2 Driver → Native Database Library → Database

Advantages

Faster than Type 1 driver

Better performance due to no ODBC layer

Disadvantages

Requires native database client libraries on client machine
Platform dependent
Not suitable for web-based applications or applets

Type 3 Driver – Network Protocol Driver (Pure Java Driver)

Description

Uses a **middleware server** between client and database
Middleware converts JDBC calls into database-specific protocol
One driver can support multiple databases

Architecture (Flow)

Java Application → JDBC API → Type 3 Driver → Middleware Server → Database

Advantages

No database-specific library required on client
Platform independent
Suitable for internet and web applications
Supports multiple databases with a single driver
Middleware can provide services like load balancing and logging

Disadvantages

Extra middleware layer may reduce performance
Increased system complexity

Type 4 Driver – Native Protocol Driver (Pure Java Driver)

Description

Converts JDBC calls **directly into database-specific protocol**
Fully written in Java
Communicates directly with database server

Architecture (Flow)

Java Application → JDBC API → Type 4 Driver → Database Server

Advantages

100% pure Java
Platform independent
Best performance
No middleware or native library required
Easy debugging and maintenance

Disadvantages

Separate driver required for each database
Database dependent

7.b. Illustrate the steps involved in connecting a java program to a database.

Seven Basic Steps in Using JDBC

1. Load the Driver

2. Define the Connection UR

3. Establish the Connection

4. Create a Statement Object

5. Execute a query

6. Process the results

7. Close the Connection

Step 0: import the java.sql package

An application that uses the jdbc API must import the java.sql package

import java.sql.*;

Step 1: Load a JDBC Driver

Prior to JDBC 4.0 it is needed to seperately load the driver and register the driver but in jdbc 4.0 it is no longer needed to

register the driver

Class.forName("sun.jdbc.odbc:jdbcodbcDriver");

Step 2: Establishing a connection

Once a driver is loaded we can establish a connection to db

Connection con= DriverManager.getconnection(dburl,username,password)

DriverManager Connects to given JDBC URL with given user name and password

A Connection represents a session with a specific database.

The connection to the database is established by getConnection(), which requests access to the database from the DBMS.

A Connection object is returned by the getConnection() if access is granted; else getConnection() throws a

SQLException.

Sometimes a DBMS requires extra information besides userID & password to grant access to the database.

This additional information is referred as properties and must be associated with Properties or Sometimes DBMS grants access

to a database to anyone without using username or password.

Ex: Connection c = DriverManager.getConnection(url) ;

Step 3: Create a statement

A statement object is needed to execute the query and obtain the results produced by it.

Statement st= con.createStatement();

Step 4: Execute the statement

The db statements can be executed by using methods like executeQuery().

executeQuery() takes querystring as an argument and returns the results as ResultSet object

ResultSet object contains the data returned by the query and the methods for retrieving the data

Ex: ResultSet rs=stmt.executeQuery("select * from employee");

Step 5: Process the result

The ResultSet consists of tuples and returns one tuple at a time when the next() is applied.

ResultSet acts as an iterator

```
While(rs.next())
{
System.out.println(rs.getString(1)+" "+rs.getInt("salary");
}
```

Getters can be used by referring position/name to retrieve the values

Step 6: Close the statement

stmt.close();

Step 7: Close the connection

Commit()

con.close()

8.a. Discuss the methods of the Statement interface are used in JDBC with an appropriate example.

The Statement object is used whenever J2EE component needs to immediately

execute a query without first having the query compiled.

Statement Object contains 3 methods:

Execute() (used for DDL commands like, Create, Alter, Drop)

executeUpdate()(Used for DML commands like, Insert, Update,

Delete)

exceuteQuery() (Used for Select command)

The execute() method is used during execution of DDL commands and also used

when there may be multiple results returned.

The executeUpdate() executes INSERT, UPDATE, DELETE, and returns an int

value specifying the number of rows affected or 0 if zero rows selected

The executeQuery() method, which passes the query as an argument. The query

is then transmitted to the DBMS for processing.

The executeQuery() □ method executes a simple select query and returns a

ResultSet object.

The ResultSet object contains rows, columns, and metadata that represent data

requested by query.

8.b. Create a Java JDBC program that demonstrates CRUD operations on a Book database table containing book_id,book_name and book_author

```
import  java.sql.*;
import  java.util.Scanner;
public class BookCRUD {

    // Database credentials
    static final String URL = "jdbc:oracle:thin:@localhost:1521:xe";
    static final String USER = "system";
    static final String PASSWORD = "oracle";

    public static void main(String[] args) {

        try {
            // Load Driver
            Class.forName("oracle.jdbc.driver.OracleDriver");

            // Establish Connection
            Connection con = DriverManager.getConnection(URL, USER, PASSWORD);

            Scanner sc = new Scanner(System.in);
            int choice;

            do {
                System.out.println("\n--- BOOK DATABASE CRUD OPERATIONS ---");
                System.out.println("1. Insert Book");
                System.out.println("2. View Books");
```

```
            System.out.println("3. Update Book");
            System.out.println("4. Delete Book");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");

            choice = sc.nextInt();

            switch (choice) {

               case 1: // INSERT
                  System.out.print("Enter Book ID: ");
                  int id = sc.nextInt();
                  System.out.print("Enter Book Name: ");
                  String name = sc.next();
                  System.out.print("Enter Book Author: ");
                  String author = sc.next();

                  String insertSQL = "INSERT INTO Book VALUES (?, ?, ?)";
                  PreparedStatement psInsert = con.prepareStatement(insertSQL);
                  psInsert.setInt(1, id);
                  psInsert.setString(2, name);
                  psInsert.setString(3, author);
                  psInsert.executeUpdate();

                  System.out.println("Book inserted successfully.");
                  break;

               case 2: // READ
                  String selectSQL = "SELECT * FROM Book";
                  Statement stmt = con.createStatement();
                  ResultSet rs = stmt.executeQuery(selectSQL);
                 System.out.println("\nBook_ID  Book_Name  Book_Author");
while (rs.next()) {
                     System.out.println(
                           rs.getInt(1) + "       " +
                           rs.getString(2) + "       " +
                           rs.getString(3));
                  }
                  break;

               case 3: // UPDATE
                  System.out.print("Enter Book ID to update: ");
                  int uid = sc.nextInt();
                  System.out.print("Enter New Book Name: ");
                  String newName = sc.next();
                  System.out.print("Enter New Author Name: ");
```

```
                String newAuthor = sc.next();

                String updateSQL = "UPDATE Book SET book_name=?, book_author=?
WHERE book_id=?";
                PreparedStatement psUpdate = con.prepareStatement(updateSQL);
                psUpdate.setString(1, newName);
                psUpdate.setString(2, newAuthor);
                psUpdate.setInt(3, uid);
                psUpdate.executeUpdate();

                System.out.println("Book updated successfully.");
                break;

            case 4: // DELETE
                System.out.print("Enter Book ID to delete: ");
                int did = sc.nextInt();

                String deleteSQL = "DELETE FROM Book WHERE book_id=?";
                PreparedStatement psDelete = con.prepareStatement(deleteSQL);
                psDelete.setInt(1, did);
                psDelete.executeUpdate();

                System.out.println("Book deleted successfully.");
                break;

            case 5:
                System.out.println("Exiting...");
                break;

            default:
                System.out.println("Invalid choice!");
          }

        } while (choice != 5);

        con.close();
        sc.close();

      } catch (Exception e) {
        e.printStackTrace();
      }
   }
}
```

9.a. Describe the architecture of MVC?

The MVC pattern separates the application into three main

components: model, view, and controller.

Model: The model represents the data of the application. It can be

anything from a simple POJO to a complex database entity.

View: The view is responsible for rendering the model data to the

user. It can be a JSP file,even a REST API

endpoint.

Controller: The controller is responsible for handling user requests

and updating the model. It can perform tasks such as validating input,

calling business logic, and redirecting the user to another page.

9.b. Explain the concept of View in the MVC architecture. Discuss its roles, responsibilities and need in web applications

In the **MVC (Model–View–Controller)** architecture, the **View** is the component responsible for presenting data to the user. It represents the **user interface (UI)** of the application and displays information received from the **Model**, as instructed by the **Controller**. The View focuses only on **how the data is shown**, not on how it is processed or stored.

Roles of the View

1. **User Interface Representation**
    - Displays application data to the user
    - Shows text, images, tables, charts, forms, etc.
2. **Data Presentation**
    - Formats raw data (from Model) into a human-readable form
    - Example: displaying dates, prices, lists of records
3. **User Interaction**
    - Accepts user inputs (clicks, form submissions)
    - Sends these inputs to the **Controller**, not directly to the Model

Responsibilities of the View

- Render the UI based on data provided
- Display error messages and success messages
- Update the UI when data changes
- Maintain separation from:
    - **Business logic** (Model)
    - **Request handling logic** (Controller)

Need for View in Web Applications

1. Separation of Concerns

- UI logic is separated from business logic

- Developers and designers can work independently

2. Easy Maintenance

- Changes in UI do not affect Model or Controller
- Same data can be shown in multiple views

3. Reusability

- One Model can be used with multiple Views
- Example: same data shown as table, chart, or report

4. Better User Experience

- Views focus purely on presentation and usability
- Easy to enhance UI using CSS, JS, frontend frameworks

5. Scalability

- Helps in building large applications
- Different views for different devices (mobile, desktop)

10.a. Explain the request-response flow handled by a controller in Spring MVC.

Imagine a web browser sends a request for a specific webpage or resource to a Spring MVC-based web application.
The web server, such as Apache Tomcat, receives this request.
The DispatcherServlet:
At the heart of Spring MVC is the DispatcherServlet. It acts as the gatekeeper.
When a request arrives, the DispatcherServlet receives the request for The "/home" URL and takes charge of routing it to the right controller.

URL Mapping
URL mapping is like a map that tells the DispatcherServlet which part of your code should handle this request.
For example, a URL like /home this could be mapped to a controller that shows the home page.

The Controller:
Once the DispatcherServlet knows where to send the request, it sends it to the right controller. Let's say there's a controller named "HomeController" for handling the home page.
The controller is like the traffic police for your web application.
It receives the request, processes it, and prepares data to show in the response.

The Model:
In Spring MVC, the data that's going to be shown on the webpage is stored in something called a "Model."

Assume in the “HomeController,” a Model is used to store data like the message “Welcome to our website!”. The controller fills this container with the necessary data.

The View:
Now, the DispatcherServlet figures out which “View” should be used to render the Model. Consider the “home.jsp” as the View for the home page.
The View is the HTML template that holds the structure of the webpage.
It’s where the data from the Model is inserted to create the final page.

The Response:
Finally, the DispatcherServlet sends the View with the filled-in Model data back to the web server.
The web server sends this page as a response to the user’s browser.
Finally, the user’s browser displays the home page with the message “Welcome to our website!”

10.b. Discuss the role of model in MVC application.
In the **MVC (Model–View–Controller)** architecture, the **Model** represents the **core of the application**. It is responsible for **managing data, business rules, and application logic**. The Model directly interacts with the **database** and performs operations required by the application, independent of how the data is displayed or how user requests are handled.

Roles and Responsibilities of Model

1. **Data Management**
    - Stores and manages application data
    - Represents domain objects and entities
2. **Business Logic Handling**
    - Implements rules, validations, and calculations
    - Ensures data integrity and consistency
3. **Database Interaction**
    - Performs CRUD operations (Create, Read, Update, Delete)
    - Communicates with databases through DAO or Repository layers
4. **Independence from UI**
    - Does not depend on View or Controller
    - Can be reused across different applications or interfaces