CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | |

Internal Assessment Test II –December 2025

Sub:	**Web Programming using Java**							**Sub Code:**	**MMCB311F**
Date:	**18/12/25**	**Duration:**	**90 min's**	**Max Marks:**	**50**	**Sem:**	**III**	**Branch:**	**MCA**

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

		MARKS	OBE	
			CO	RBT
	PART I			
1	Illustrate the steps involved in connecting a java program to a database with code snippets.	[10]	CO4	L3
	OR			
2	Describe DOM structure in Javascript.	[10]	CO1	L1
	PART II			
3	List and explain different JDBC Driver types along with its advantages and disadvantages.	[10]	CO4	L2
	OR			
4	Discuss the role of model in MVC application	[10]	CO5	L2
	PART III			
5	Explain how do you handle button events in JavaScript.	[10]	CO1	L2
	OR			
6	Develop a program using JDBC to perform CRUD operations into Book database. Table consists of Book_idint(5), book_name varchar2(20), book_author varchar2(20).	[10]	CO4	L4
	PART IV			
7	Describe the architecture of MVC?	[10]	CO5	L2
	OR			
8	Explain PreparedStatement and CallableStatement? Explain with Code Snippets.	[10]	CO4	L2
	PARTV			
9	Describe HTTP Response Headers.	[10]	CO1	L1
	OR			
10	Write a JDBC program to update and delete a row from ResultSet.	[10]	CO4	L4

1. Illustrate the steps involved in connecting a java program to a database with code snippets.

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 separately 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();

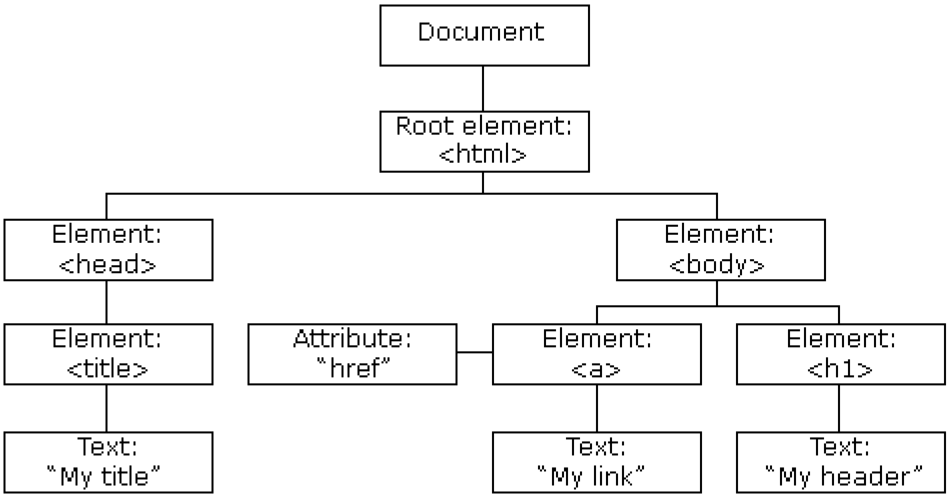
Step 7: Close the connection

Commit()

con.close()

2. Describe DOM structure in JavaScript.

The **Document Object Model (DOM)** is a programming interface that represents an HTML document as a **hierarchical tree structure**. When a web page is loaded, the browser parses the HTML code and converts it into a DOM, which allows JavaScript to dynamically access and manipulate the content, structure, and style of the web page.



At the top of the DOM hierarchy is the **Document node**, represented by the `document` object. It acts as the entry point for accessing all elements in the web page. The Document node contains the **root element**, which is the `<html>` element. The `<html>` element further branches into two main child elements: `<head>` and `<body>`.

The `<head>` element contains metadata such as the page title, links to stylesheets, and scripts, while the `<body>` element contains the visible content of the web page such as headings, paragraphs, images, and forms. Each HTML tag is represented as an **element node**, and the text inside an element is represented as a **text node**. Attributes like `id`, `class`, and `href` are represented as **attribute nodes**.

The DOM follows **parent–child–sibling relationships**. An element that contains other elements is called a parent, the contained elements are children, and elements at the same level are siblings. This tree-based structure enables JavaScript to traverse the document efficiently.

JavaScript provides several properties and methods to access and navigate the DOM, such as `document.documentElement`, `document.body`, `parentNode`, `childNodes`, and `nextSibling`. Using these, developers can dynamically modify HTML elements, change styles, add or remove elements, and respond to user events.

3. 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

4. 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

o Stores and manages application data

o Represents domain objects and entities

2. Business Logic Handling

o Implements rules, validations, and calculations

o Ensures data integrity and consistency

3. Database Interaction

o Performs CRUD operations (Create, Read, Update, Delete)

o Communicates with databases through DAO or Repository layers

4. Independence from UI

o Does not depend on View or Controller

o Can be reused across different applications or interfaces

5. Explain how do you handle button events in java script.

Button events in JavaScript are handled using the **event handling mechanism**, which allows a web page to respond to user actions such as clicking, double-clicking, or hovering over a button. Event handling is a key concept in creating **interactive and dynamic web applications**.

When a user interacts with a button, the browser generates an **event**. JavaScript listens for this event and executes a specific block of code called an **event handler**.

1. Inline Event Handling

In this method, the event handler is written directly inside the HTML element using event attributes.

```
<button onclick="displayMessage()">Click</button>

<script>
function displayMessage() {
    alert("Button clicked");
}
</script>
```

Here, the `onclick` attribute triggers the function when the button is clicked.

2. Event Handler Property

JavaScript allows assigning an event handler using element properties.

```
<button id="btn">Click</button>

<script>
document.getElementById("btn").onclick = function () {
    alert("Button clicked");
```

```
};
</script>
```

In this method, only one event handler can be attached to the button at a time.

3. Using addEventListener() Method

This is the **recommended and modern approach** to handle button events.

```
<button id="btn">Click</button>

<script>
document.getElementById("btn").addEventListener("click", function () {
    alert("Button clicked");
});
</script>
```

The `addEventListener()` method allows multiple event handlers for the same button and keeps JavaScript separate from HTML.

4. Event Object

When an event occurs, JavaScript passes an **event object** to the handler, which contains information about the event such as event type and target element.

```
document.getElementById("btn").addEventListener("click", function(event) {
    console.log(event.type);
});
```

6. Develop a program using JDBC to perform CRUD operations into BOOK database. Table consists of Book_id int(5),book_name varchar2(20), book_author varchar2(20)

```
import java.sql.*;

import java.util.Scanner;


public class BookCRUD {


   static final String URL = "jdbc:oracle:thin:@localhost:1521:xe";

   static final String USER = "system";

   static final String PASS = "password";
```

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    try {
        // 1. Load Driver
        Class.forName("oracle.jdbc.driver.OracleDriver");

        // 2. Establish Connection
        Connection con = DriverManager.getConnection(URL, USER, PASS);

        while (true) {
            System.out.println("\n1.Insert  2.View  3.Update  4.Delete  5.Exit");
            System.out.print("Enter choice: ");
            int choice = sc.nextInt();

            switch (choice) {

                // CREATE
                case 1:
                    String insertSQL = "INSERT INTO BOOK VALUES (?,?,?)";
                    PreparedStatement psInsert = con.prepareStatement(insertSQL);

                    System.out.print("Enter Book ID: ");
                    psInsert.setInt(1, sc.nextInt());
                    System.out.print("Enter Book Name: ");
                    psInsert.setString(2, sc.next());
                    System.out.print("Enter Author Name: ");
                    psInsert.setString(3, sc.next());
```

```
                psInsert.executeUpdate();
                System.out.println("Book Inserted Successfully");
                break;

            // READ
            case 2:
                String selectSQL = "SELECT * FROM BOOK";
                Statement stmt = con.createStatement();
                ResultSet rs = stmt.executeQuery(selectSQL);

                System.out.println("Book_id\tBook_Name\tAuthor");
                while (rs.next()) {
                    System.out.println(
                            rs.getInt(1) + "\t" +
                            rs.getString(2) + "\t" +
                            rs.getString(3)
                    );
                }
                break;

            // UPDATE
            case 3:
                String updateSQL = "UPDATE BOOK SET book_name=?, book_author=? WHERE
Book_id=?";
                PreparedStatement psUpdate = con.prepareStatement(updateSQL);

                System.out.print("Enter New Book Name: ");
                psUpdate.setString(1, sc.next());
                System.out.print("Enter New Author Name: ");
                psUpdate.setString(2, sc.next());
```

```
            System.out.print("Enter Book ID to Update: ");
            psUpdate.setInt(3, sc.nextInt());

            psUpdate.executeUpdate();
            System.out.println("Book Updated Successfully");
            break;

        // DELETE
        case 4:
            String deleteSQL = "DELETE FROM BOOK WHERE Book_id=?";
            PreparedStatement psDelete = con.prepareStatement(deleteSQL);

            System.out.print("Enter Book ID to Delete: ");
            psDelete.setInt(1, sc.nextInt());

            psDelete.executeUpdate();
            System.out.println("Book Deleted Successfully");
            break;

        case 5:
            con.close();
            System.out.println("Connection Closed");
            System.exit(0);

        default:
            System.out.println("Invalid Choice");
    }
}
```

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

7. 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.

8. Explain Prepared Statement and Callable Statement? Explain with Code Snippets.

The prepared Statement object allows you to execute parameterized queries. A SQL query can be precompiled and executed by using the Prepared Statement object. ·

Ex: Select * from publishers where pub_id=?

Here a query is created as usual, but a question mark is used as a placeholder for a value· that is inserted into the query after the query is compiled.

The preparedStatement() method of Connection object is called to return the preparedStatement object.

Ex:

```
import java.sql.*;
public class JdbcDemo {
```

```
publicstaticvoidmain(Stringargs[]){ try{

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Connection con=DriverManager.getConnection("jdbc:odbc:MyDataSource","khutub","");

PreparedStatement pstmt;

pstmt=con.prepareStatement("select*fromemployeewhereUserName=?");
pstmt.setString(1,"khutub");

ResultSetrs1=pstmt.executeQuery(); while(rs1.next()){

System.out.println(rs1.getString(2));

}

} // end of try

catch(Exception e){System.out.println("exception"); }

} //end of main

} // end of class
```

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 type and style of code depends on the DBMS vendor and can be written in PL/SQL, Transact-SQL, C, or other programming languages.

IN, OUT and INOUT are the three parameters used by the CallableStatement object to call a stored procedure.

The IN parameter contains any data that needs to be passed to the stored procedure and whose value is assigned using the setxxx() method.

The OUT parameter contains the value returned by the stored procedures. The OUT parameters must be registered using the registerOutParameter() method, later retrieved by using the getxxx()

The INOUT parameter is a single parameter that is used to pass information to the stored procedure and retrieve information from the stored procedure.

Ex:

```
Connection con;  try{

String query = "{CALL LastOrderNumber(?))}";

CallableStatement stat = con.prepareCall(query);  stat.registerOutParameter( 1
,Types.VARCHAR);  stat.execute();

String lastOrderNumber = stat.getString(1);  stat.close();
```

```
}
catch (Exception e){}
```

9. Describe HTTP Response Headers.

1. Content-Type

- Specifies the media type of the response body.
  `Content-Type: text/html`

2. Content-Length

- Indicates the size of the response body in bytes.

```
Content-Length: 1024
```

3. Server

- Identifies the web server software.
  `Server: Apache/2.4.41`

4. Date

- Shows the date and time when the response was generated.

```
Date: Wed, 14 Aug 2025 10:30:00 GMT
```

5. Cache-Control

- Controls caching behavior of the response.

```
Cache-Control: no-cache
```

6. Expires

- Specifies the expiration time of the cached response.

```
Expires: Thu, 15 Aug 2025 10:30:00 GMT
```

7. Set-Cookie

- Sends a cookie from server to client.

```
Set-Cookie: sessionId=abc123; HttpOnly
```

8. Location

- Used in redirection responses (3xx status codes).

```
Location: https://www.example.com/home
```

9. Access-Control-Allow-Origin

- Specifies allowed origins for CORS.

```
Access-Control-Allow-Origin: *
```

10. WWW-Authenticate

- Used for authentication challenges.

```
WWW-Authenticate: Basic realm="User Area"
```

In Java web applications, response headers can be set using:

```
response.setHeader("Content-Type", "text/html");
```

10. Write a jdbc program to update and delete a row from resultset.

```
import java.sql.*;


public class UpdateDeleteResultSet {


    public static void main(String[] args) {


        try {
            // 1. Load Driver
            Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
// 2. Establish Connection
Connection con = DriverManager.getConnection(
        "jdbc:oracle:thin:@localhost:1521:xe",
        "system",
        "password");

// 3. Create Updatable ResultSet
Statement stmt = con.createStatement(
        ResultSet.TYPE_SCROLL_INSENSITIVE,
        ResultSet.CONCUR_UPDATABLE);

// 4. Execute Query
ResultSet rs = stmt.executeQuery("SELECT * FROM BOOK");

// -------- UPDATE OPERATION --------
while (rs.next()) {
    if (rs.getInt("Book_id") == 101) {
        rs.updateString("book_name", "Java Programming");
        rs.updateString("book_author", "James Gosling");
        rs.updateRow();   // commit update
        System.out.println("Record Updated Successfully");
    }
}

// Move cursor back to start
rs.beforeFirst();

// -------- DELETE OPERATION --------
while (rs.next()) {
```

```
            if (rs.getInt("Book_id") == 102) {
                rs.deleteRow();  // delete current row
                System.out.println("Record Deleted Successfully");
            }
        }

        // 5. Close Connection
        rs.close();
        stmt.close();
        con.close();

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