Assignment - I

Date
Page 01

Q-1 Write a program to demonstrate the use of JDBC transactions.

-> A JDBC transaction is a sequence of SQL statements that are executed as a single unit of work. By default, each SQL statement in JDBC is auto-committed, meaning changes are immediately saved to the database. However, for transactional consistency, we can disable auto-commit and manually control commits and rollbacks.

*Code :-

```
import java.sql.Connection;
import javal.sql.DriverManager;
import java.sql.SQLException;

public class Main {

    private static final String DB_URL
        = "jdbc:mysql://localhost:3306/mca";
```

```
private static final String DB_USERNAME
        = "root";
private static final String DB_PASSWORD
        = "";

public static void main(String[] args){
  Connection connection = null;

  try{
    connection = DriverManager.getConnection
    (DB_URL,DB_USERNAME,DB_PASSWORD);

    if(connection != null){
      System.out.println("Connection to
          the database is successfull!");
    }else{
      System.out.println("DB Not connected");
    }

  }catch(SQLException e){
    e.printStackTrace();
  }finally{
```

```
try {
    if (connection != null) {
        connection.close();
        System.out.println("Connection
                            Close");
    }
} catch (SQLException e) {
    e.printStackTrace();
}
}
}
}
```

Q-2 Write a program to call a stored procedure using CallableStatement in JDBC

-> A stored procedure is a precompiled set of SQL statements stored in the database. It allows logic to be executed on the database side, improving performance and security.

* Example

Step 1 : Create a Stored Procedure in MySQL

```
DELIMITER //
CREAT PROCEDURE GetEmployeeSalary
        (IN empId INT, OUT empSalary DOUBLE)
BEGIN
    SELECT salary INTO empSalary FROM
        employees WHERE id = empId;
END //
DELIMITER ;
```

Step 2 : Call Stored Procedure in JDBC

```
import java.sql.*;

public class CallableStatementExample {

    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/
                      testdb";
        String user = "root";
        String password = "password";

        try (Connection conn = DriverManager.
                getConnection(url, user, password)) {
            CallableStatement cs = conn.prepareCall
                ("{CALL GetEmployeeSalary(?, ?)}");
            cs.setInt(1, 101);
            cs.registerOutParameter(2, Types.DOUBLE);
            cs.execute();

            double salary = cs.getDouble(2);
            System.out.println("Employee Salary: "+salary);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Q-3 What are cookies, and how can servlets use them to maintain client state?

-> Cookies are small pieces of data stored on the client's browser to track user sessions.

a) Creating and Sending Cookies:

```
Cookie cookie = new Cookie("username",
                           "JohnDoe");
cookie.setMaxAge(3600);
respone.addCookie(cookie);
```

b) Retrieving Cookies in Servlet:

```
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for (Cookie c : cookies) {
        if (c.getName().equals("username")){
            System.out.println("User: " + c.getValue());
        }
    }
}
```

Q-4 What are the differences between PrintWriter and ServletOutputStream when generating a response?

* PrintWriter (response.getWriter())

- Used for writing character-based content (text, HTML, JSON).
- Works well for sending simple web pages or structured textual data.
- Cannot handle binary data like images, PDFs, or file downloads.

* Example :-

```
PrintWriter out = response.getWriter();
out.println("<h1>Hello, World!</h1>");
```

- Throws IllegalStateException if getOutputStream() was already called.

* ServletOutputStream (response.getOutputStream())

- Used for writing binary data (images, videos, PDFs, file downloads).
- Can be used to send raw bytes over HTTP

*Example

```
ServletOutputStream out = response.getOutputStream();
byte [] imageData = getImageBytes();
out.write(imageData);
```

- Throws IllegalStateException if getWriter() was already called.

Feature	PrintWriter (getWriter())	ServletOutputStream (getOutputStream())
Type of Data	Character / Textual Data	Binary Data (Images, PDFs, Videos)
Data Format	HTML, JSON, Plaintext.	Byte Streams (files, media)
Methods Used	println(), write(String)	write(byte[]), flush(), close()
Example Usage	Generating HTML response	Downloading an image or file.
Handling Character Encoding	Uses character encoding like UTF-8	Works with raw bytes, no encoding needed.
Throws Exception	IllegalStateException if getOutputStream() is used first	IllegalStateException if getWriter() is used first.

Q-5 Explain the basic structure of a servlet program with an example.

-> A Servlet is a Java class that extends the capabilities of a web server by handling HTTP requests and generating dynamic web content. It runs on a Servlet Container and follows a specific life cycle.

* Basic Structure

1) Import Required Packages
2) Extend HttpServlet Class
3) Override doGet() and doPost() Methods
4) Set Response Content Type
5) Write Response to ~~Cline~~ Client

* Code.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet {
    public void doGet
        (HttpServletRequest, request, HttpServletRespons response)
    throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello, Servlet World!</h1>");
    }
}
```

Q-6 Explain the servlet life cycle with the role of the following methods:
- init()
- service()
- destroy()

-> A Servlet is a Java class that runs on a web server to handle client requests dynamically. The servlet life cycle consists of three main stages.

1) Initialization (init())
-> Called once when the servlet is loaded.

2) Request Handling (service())
-> Called multiple times for each client request

3) Destruction (destroy())
-> Called once before the servlet is removed from memory.

1) Initialization (init())

- Called once when the servlet is first loaded into memory
- Used for setting up resources like database connections, logging, and configurations.

* Example :-

```
public void init() throws ServletException {
    DatabaseConnection.initialize();
}
```

2) Request Handling (service())

- Called each time the servlet receives a request
- Determines the request type (GET, POST, etc.) and calls doGet() or doPost().

* Example :-

~~prote~~

```
protected void doGet (HttpServletRequest request,
        HttpServletRespone response)
        throws ServletException, IOException {
    response.getWriter().println("Processing Get
                                  request...");
}
```

3] Destruction (destroy())

- Called once before the servlet is removed from memory.
- Used to release resources such as closing database connections or stopping background tasks.

* Example :-

```
public void destroy() {
        DatabaseConnection.close();
}
```

Q-7 What does Class.forName() method do?

-> Class.forName(String className) is a Java Reflection API method used to dynamically load a class at runtime. It is commonly used in JDBC and other frameworks where class names are provided as strings.

* Working of Class.forName()

- It loads the specified class into memory
- It initializes the class by executing its static blocks
- It returns a Class object that represents the loaded class.

* Example.

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

Q-8 Explain the scenario when you will use ResultSetMetadata Class with example.

-> ResultSetMetadata is an interface in JDBC used to retrieve metadata (structure information) about a ResultSet. It provides details about the columns of a database query result, such as column names, data types, and sizes.

-> When writing generic code that works with different tables without knowing the schema, ResultSetMetaData helps retrieve column details dynamically.

For Example :-

- You have a query that can return results from any table, but you don't know the column names beforehand.

- ResultSetMetadata helps in printing all column names and their data types.

* Example :-

```
import java.sql.*;

public class ResultSetMetaDataExample {
    public static void main(String[] args) {
        try {

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

        Connection con = DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/mydb",
             "root", "password");

        Statement stmt = con.createStatement();
        ResultSet rs = stmt.executeQuery
                        ("SELECT * FROM employees");

        ResultSetMetaData rsmd = rs.getMetaData();
        int columnCount = rsmd.getColumnCount();

        for(int i=1; i<=columnCount; i++) {
```

```
            System.out.println("Column" + i + ": " +
                    rsmd.getColumnName(i) +
                "(Type: " + rsmd.getColumn
                TypeName(i) + ")");
        }
        rs.close();
        stmt.close();
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

Q-9 Why get() and post() when implemented in Servlet throws ServletException and IO Exception ?

-> When implementing doGet() and doPost() methods in a servlet, we must declare that they throw ServletException and IOException.

*Code:-

```
protected void doGet (HttpServletRequest
        request, HttpServletResponse response)
        throws ServletException, IOException {
        // Code to handle GET request.
}

protected void doPost (HttpServletRequest
        request, HttpServletResponse respones)
        throws ServletException, IOException {
        // Code to handle POST request
}
```

11 ServletException - Issues Related to Servlet Processing

- ServletException occurs when there is a logical error or problem with servlet execution
- It indicates problems that the servlet cannot recover from

*Example :-

```
throw new ServletException("Servlet is not properly configured");
```

2] IOException - Issues Related to Input / Output Operations

- IOException occurs when there are problems with input or output operations, typically related to network or stream handling.

- It happens when:
  -> The client connection is lost before sending a response
  -> There is an issue reading data from a request
  -> Writing to the response output stream fails due to network problems

*Example :-

```
PrintWriter out = response.getWriter();
out.println("Hello, Client!");
out.close();
```

10. Explain the scenarios when execute(), executeUpdate() and executeQuery() must be used with example.

-> JDBC provides three diffrent methods in the Statement and PreparedStatement interfaces to execute SQL queries:

1] Using executeQuery()

- Used when fetching data from the database
- Returns a ResultSet containing query results.

* Code

```
import java.sql.*;

public class ExecuteQueryExample {
    public static void main(String[] args){
        try {
```

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

Connection con = DriverManager.getConnection
  ("jdbc:mysql://localhost:3306/mydb", "root",
   "password");

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery
                ("SELECT * FROM employees");
while (rs.next()) {
    System.out.println("ID: " + rs.getInt("id")
    + ", Name: " + rs.getString("name"));
}
  rs.close();
  stmt.close();
  con.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}
```

2] Using executeUpdate()

- Used for modifying data
  (INSERT, UPDATE, DELETE).
- Used for database structure changes
  (CREATE, ALTER, DROP).
- Returns an integer representing the number of affected rows.

* Example

```
Statement stmt = con.createStatement();
int rowsAffected = stmt.executeUpdate
        ("INSERT INTO employees (name, salary)
          VALUES ('John Doe', 5000)");

System.out.println("Rows Inserted: " +
                   rowsAffected);
```

3] Using execute()

- Used when you don't know if the query will return a ResultSet or an update count.

* Example :-

```
Statement stmt = con.createStatement();

boolean hasResultSet = stmt.execute("SELECT *
                                    FROM employees");

if (hasResultSet) {
    ResultSet rs = stmt.getResultSet();
    while (rs.next()) {
        System.out.println("ID: " + rs.getInt("id")
                           + ", Name : " + rs.getString("name"));
    }
    rs.close();
} else {
    int updateCount = stmt.getUpdateCount();
    System.out.println("Rows affected: " + updateCount);
}
```