Unit 13: Database Programming using JDBC [2 Hrs.]

1. Definition of JDBC

Definition: JDBC (Java Database Connectivity) is a Java API that provides a standard way for Java programs to interact with relational databases. It allows us to execute SQL queries, retrieve results, and perform database operations like insert, update, and delete.

Key Components of JDBC:

1. **DriverManager**: Manages database drivers and establishes a connection to the database.
2. **Connection**: Represents a connection to the database.
3. **Statement**: Used to execute SQL queries.
4. **ResultSet**: Represents the result of a query (e.g., rows returned by a `SELECT` statement).

---

2. Steps to Connect to a Database using JDBC

To work with JDBC, we follow these steps:

1. **Load the JDBC Driver**: Register the database driver.
2. **Establish a Connection**: Use `DriverManager` to connect to the database.
3. **Create a Statement**: Use the `Connection` object to create a `Statement` or `PreparedStatement`.
4. **Execute Queries**: Execute SQL queries using the `Statement` object.
5. **Process Results**: Use the `ResultSet` object to process query results.
6. **Close Resources**: Close the `Connection`, `Statement`, and `ResultSet` objects to release resources.

---

3. Using `Connection`, `Statement`, and `ResultSet` Interfaces

a. Connection Interface

Definition: The `Connection` interface represents a connection to the database. It is used to create `Statement` objects and manage transactions.

1. **Download the MySQL Connector/J (JDBC Driver):** We need to download the MySQL JDBC driver (`mysql-connector-java`).
   - Visit the official MySQL website to download the JDBC driver: MySQL Connector/J
2. **Add the JDBC Driver to Project Classpath:**
   - After downloading the `.jar` file (for example, `mysql-connector-java-x.x.x.jar`), copy it to project directory.
   - In IDE (such as IntelliJ IDEA), add the `.jar` file to the project's classpath:
     - Right-click on the project in IDE.
     - Go to **Module Settings** (for IntelliJ IDEA, it's under `File` -> `Project Structure` -> `Modules`).
     - In the **Dependencies** tab, click the **+** button and add the `.jar` file.

3. **Rebuild Project:** After adding the JDBC driver, rebuild project. In IntelliJ IDEA, click on `Build` -> `Rebuild Project` .

Lab 1: Connection Interface:

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

public class ConnectionExample {
    public static void main(String[] args) {
        Connection connection = null;
        try {
            // Step 1: Load the JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Step 2: Define the database URL, username, and password
            String url = "jdbc:mysql://localhost:3306/mydatabase";
            String username = "root";
            String password = "password";

            // Step 3: Establish a connection to the database
            connection = DriverManager.getConnection(url, username, password);

            // Step 4: Confirm successful connection
            System.out.println("Successfully connected to the database!");

        } catch (ClassNotFoundException e) {
            // Handle JDBC driver loading errors
            System.out.println("JDBC driver not found.");
            e.printStackTrace();
        } catch (SQLException e) {
            // Handle database connection errors
            System.out.println("Failed to connect to the database.");
            e.printStackTrace();
        } finally {
            // Step 5: Close the connection if it was established
            if (connection != null) {
                try {
                    connection.close();
                    System.out.println("Connection closed.");
                } catch (SQLException e) {
                    System.out.println("Failed to close the connection.");
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Explanation:

- We load the MySQL JDBC driver using `Class.forName()` .

- We establish a connection to the database using `DriverManager.getConnection()`.
- Finally, we close the connection to release resources.

Sample Output:

```
Successfully connected to the database!
```

---

b. Statement Interface

Definition: The `Statement` interface is used to execute SQL queries (e.g., `SELECT`, `INSERT`, `UPDATE`, `DELETE`).

```sql
CREATE TABLE employees (
    id INT PRIMARY KEY,                  -- Unique identifier for each employee
    name VARCHAR(100) NOT NULL,          -- Employee's name (up to 100 characters)
    salary DECIMAL(10, 2) NOT NULL       -- Employee's salary (e.g., 50000.00)
);
```

Lab 2: Statement Interface:

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

public class StatementExample {
    public static void main(String[] args) {
        Connection connection = null;
        Statement statement = null;
        try {
            // Step 1: Load the JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Step 2: Establish a connection to the database
            String url = "jdbc:mysql://localhost:3306/mydatabase";
//mydatabase=whitefield
            String username = "root";
            String password = "password";   //password="";
            connection = DriverManager.getConnection(url, username, password);

            // Step 3: Create a Statement object
            statement = connection.createStatement();

            // Step 4: Execute an SQL query
            String sql = "INSERT INTO employees (id, name, salary) VALUES (1, 'Sharat
Maharjan', 50000)";
            int rowsAffected = statement.executeUpdate(sql);   //executeUpdate() for
operations that modify the database (INSERT, UPDATE, DELETE)
            System.out.println(rowsAffected + " row(s) inserted successfully.");

        } catch (ClassNotFoundException e) {
            // Handle JDBC driver loading errors
            System.out.println("JDBC driver not found.");
            e.printStackTrace();
        } catch (SQLException e) {
            // Handle database connection or query execution errors
            System.out.println("Error executing SQL query.");
            e.printStackTrace();
        } finally {
            // Step 5: Close resources
            try {
                if (statement != null) {
                    statement.close();
                    System.out.println("Statement closed.");
                }
                if (connection != null) {
                    connection.close();
                    System.out.println("Connection closed.");
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources.");
                e.printStackTrace();
            }
        }
    }
}
```

Explanation:

- We create a `Statement` object using `connection.createStatement()`.
- We execute an `INSERT` query using `statement.executeUpdate()`.
- The `executeUpdate()` method returns the number of rows affected.

Sample Output:

```
1 row(s) inserted successfully.
```

---

c. ResultSet Interface

Definition: The `ResultSet` interface represents the result of a `SELECT` query. It allows us to iterate through the rows returned by the query.

Lab 3: ResultSet Interface:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class ResultSetExample {
    public static void main(String[] args) {
        Connection connection = null;
        Statement statement = null;
        ResultSet resultSet = null;
        try {
            // Step 1: Load the JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Step 2: Establish a connection to the database
            String url = "jdbc:mysql://localhost:3306/mydatabase";
            String username = "root";
            String password = "password";
            connection = DriverManager.getConnection(url, username, password);

            // Step 3: Create a Statement object
            statement = connection.createStatement();

            // Step 4: Execute a SELECT query
            String sql = "SELECT id, name, salary FROM employees";
            resultSet = statement.executeQuery(sql);   //executeQuery() for operations
that retrieve data (SELECT)

            // Step 5: Process the ResultSet
            System.out.println("Employee Details:");
            while (resultSet.next()) {
                int id = resultSet.getInt("id");
                String name = resultSet.getString("name");
                double salary = resultSet.getDouble("salary");
                System.out.println("ID: " + id + ", Name: " + name + ", Salary: " +
salary);
            }

        } catch (ClassNotFoundException e) {
            // Handle JDBC driver loading errors
            System.out.println("JDBC driver not found.");
            e.printStackTrace();
        } catch (SQLException e) {
            // Handle database connection or query execution errors
            System.out.println("Error executing SQL query.");
            e.printStackTrace();
        } finally {
            // Step 6: Close resources
            try {
                if (resultSet != null) {
                    resultSet.close();
                    System.out.println("ResultSet closed.");
                }
                if (statement != null) {
                    statement.close();
                    System.out.println("Statement closed.");
                }
                if (connection != null) {
                    connection.close();
                    System.out.println("Connection closed.");
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources.");
                e.printStackTrace();
            }
        }
    }
}
```

Explanation:

- We execute a `SELECT` query using `statement.executeQuery()`.
- We use the `ResultSet` object to iterate through the rows and retrieve column values using methods like `getInt()`, `getString()`, and `getDouble()`.

Sample Output:

```
ID: 1, Name: Sharat Maharjan, Salary: 50000.0
```

---

4. Differences Between `Statement` and `PreparedStatement`

Feature	Statement	PreparedStatement
SQL Injection	Vulnerable to SQL injection.	Prevents SQL injection.
Performance	Slower for repeated queries.	Faster for repeated queries.
Usage	Used for static SQL queries.	Used for dynamic SQL queries.

---

5. Summary

In this unit, we learned about **JDBC (Java Database Connectivity)**, which is a standard API for connecting Java applications to relational databases. We explored the following key interfaces:

1. **Connection**: Represents a connection to the database.
2. **Statement**: Used to execute SQL queries.
3. **ResultSet**: Represents the result of a query.

We also learned how to:

- Load the JDBC driver.
- Establish a connection to the database.
- Execute SQL queries using `Statement` and `PreparedStatement`.
- Process query results using `ResultSet`.