**Chapter 1: FOUNDATIONS**

**Topic – 1: Introduction To JDBC**

* **JDBC** stands for **Java Database Connection**.
* **ODBC** stands for **Oracle Database Connection**.
* Works on various **relational databases**, **non-relational databases**, **spreadsheets** & **flat files** etc.
* **JDBC** driver might be required to be changed.
* Packaged provided are ***java.sql*** & ***javax.sql***.

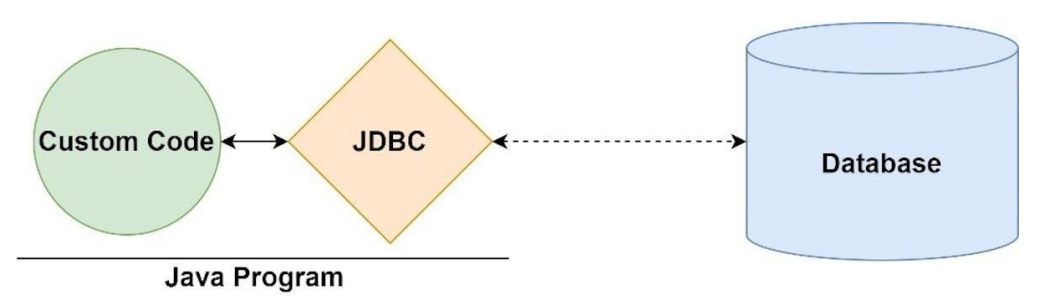
**Topic – 2: Components Of JDBC**

**The 4 Components**

* **JDBC API:** Contains **Java SE** & **Java EE** platform services with ***java.sql*** package.
* **JDBC driver manager:** Loads database driver & establishes connection among the database & driver.
* **JDBC test suite:** Tests operations like **insertion**, **deletion** & **updation**.
* **JDBC-ODBC bridge drivers:** Translates embedded database commands by **JDBC** to **ODBC**.

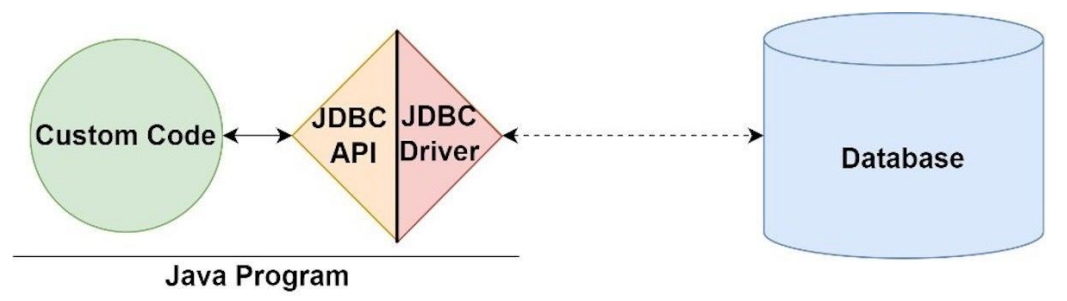
**Topic – 3: Working & Architecture**

**Working**



**Architecture**

* **JDBC API** & **JDBC driver** are two layers in **JDBC**.



**Topic – 4: Drivers**

**Introduction**

* **Drivers** are installed on user **machine adapters**.
* These convert Java programs to understandable protocols for **DBMS**.

**Types Of Drivers**

* JDBC-ODBC bridge driver (Type-1 driver)
* Native-API driver (Type-2 driver)
* Network protocol driver (Type-3 driver)
* Thin driver (Type-4 driver)

**JDBC-ODBC Bridge Driver**

* Connects driver to **ODBC**.
* **Universal driver** for ability to connect to any database.
* Needs to installed separately from **JDK**.

**Native-API Driver**

* Converts **JDBC** method calls to native database commands.
* Data transmission is comparatively more secured for using own **local API**.

**Network Protocol Driver**

* Converts **JDBC** method calls into **vendor-specific** protocols.
* No separate **installation** is required.
* Easy database migration.

**Thin Driver**

* Also called native-protocol driver.
* **Doesn’t** require any native database library, pre-programmed.

**Topic – 5: JDBC API Classes & Interfaces**

**DriverManager Class**

* ***registerDriver(Driver driver)***
* ***deregisterDriver(Driver driver)***
* ***getConnection(String Url)***
* ***getConnection(String Url, String username, String password)***

**Connection Interface**

* ***createStatement()***
* ***createStatement(intresultSetType,intr esultSetConcurrency)***
* ***setAutoCommit(boolean status)***
* ***commit()***
* ***rollback()***
* ***close()***

**Statement Interface**

* ***ResultSetexecuteQuery(String sql)***
* ***intexecuteUpdate(String sql)***
* ***execute(String sql)***
* ***executeBatch()***

**ResultSet Interface**

**Note!**

**🡪 Maintains pointers on rows.**

* ***next()***
* ***previous()***
* ***first()***
* ***booleanlast()***
* ***booleanabsolute(int row)***
* ***booleanrelative(int row)***
* ***intgetInt(intcolumnIndex)***
* ***intgetInt(String columnName)***
* ***StringgetString(intcolumnIndex)***
* ***StringgetString(StringcolumnIndex)***

**ResultSet Type Values**

* **TYPE\_FORWARD\_ONLY:** **Can’t** jump through rows.
* **TYPE\_SCROLL\_INSENSITIVE:** Can jump but **insensitive** to changes in DB.
* **TYPE\_SCROLL\_SENSITIVE:** Can jump but **sensitive** to changes in DB.

**ResultSet Concurrency Values**

* **CONCUR\_READ\_ONLY:** Result **can’t** be used to **update** the database.
* **CONCUR\_UPDATABLE:** Result can be used to **update** the database.

**Example Code**

***try {***

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

***conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/myData", "root", "pwd");***

***Statement stmt = conn.createStatement();***

***ResultSet result = stmt.executeQuery(“select \* from employees”);***

***while(result.next())***

***{***

***System.out.println(result.getInt(1) + "\t" + result.getString(2) + " " + result.getString(3));***

***}***

***conn.close();***

***}***

***catch (Exception e)***

***{***

***System.out.println("my Error :" + e.getMessage());***

***}***

**PreparedStatement Interface**

* ***setInt(intparamIndex, int value)***
* ***setString(intparamIndex, String value)***
* ***setFloat(intparamIndex, float value)***
* ***setDouble(intparamIndex, double value)***
* ***intexecuteUpdate()***
* ***ResultSetexecuteQuery()***

**Statement v/s Prepared Statement**

|  |  |
| --- | --- |
| **Statement** | **Prepared Statement** |
| **Used for static SQL statements.** | **Used for dynamic SQL statements.** |
| **Base interface.** | **Extends statement interface.** |
| **Poor performance, parsed & compiled during execution.** | **Good performance, pre-compiled codes.** |
| **Inputs are executable codes.** | **Inputs are treated as arguments.** |
| **Vulnerable to SQL injections.** | **Safe from SQL injections.** |
| **Can pass concatenated statements.** | **No need to pass concatenated statements.** |
| **Can’t read statements as binary data.** | **Can read statements as binary data.** |
| **Doesn’t use binary protocols.** | **Uses binary protocols.** |

**Example Code – II**

***import java.sql.Connection;***

***import java.sql.DriverManager;***

***import java.sql.PreparedStatement;***

***import java.util.Scanner;***

***public class Main {***

***public static void main(String[] args) {***

***int mEmpno, mDeptno;***

***String mFname, mLname, mMobile, mEmail, mJob;***

***Scanner scn = new Scanner(System.in);***

***System.out.print("Enter the valid employee number: ");***

***mEmpno = scn.nextInt();***

***System.out.print("Enter the valid employee fname: ");***

***mFname = scn.next();***

***System.out.print("Enter the valid employee lname: ");***

***mLname = scn.next();***

***System.out.print("Enter the valid employee mobile: ");***

***mMobile = scn.next();***

***System.out.print("Enter the valid employee email: ");***

***mEmail = scn.next();***

***System.out.print("Enter the valid employee job: ");***

***mJob = scn.next();***

***System.out.print("Enter the valid employee deptno: ");***

***mDeptno = scn.nextInt();***

***Connection conn = null;***

***try {***

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

***conn = DriverManager.getConnection(***

***"jdbc:mysql://localhost:3306/classicmodels?useSSL=false",***

***"root",***

***"password"***

***);***

***String SQL = "insert into empmast values(?,?,?,?,?,?,?)";***

***PreparedStatement pstmt = conn.prepareStatement(SQL);***

***// Set the value to each parameter***

***pstmt.setInt(1, mEmpno);***

***pstmt.setString(2, mFname);***

***pstmt.setString(3, mLname);***

***pstmt.setString(4, mMobile);***

***pstmt.setString(5, mEmail);***

***pstmt.setString(6, mJob);***

***pstmt.setInt(7, mDeptno);***

***int rowaffected = pstmt.executeUpdate();***

***if (rowaffected > 0) {***

***System.out.println("Record inserted successfully!");***

***} else {***

***System.out.println("Error in insert!");***

***}***

***conn.close();***

***} catch (Exception e) {***

***System.out.println("Error: " + e.getMessage());***

***}***

***}***

***}***

**ResultSetMetaData Interface**

* ***getColumnCount()***
* ***getColumnName(int index)***
* ***getColumnTypeName(int index)***

**DatabaseMetaData Interface**

* ***getDriverName()***
* ***getDriverVersion()***
* ***getUserName()***
* ***getDatabaseProductName()***
* ***getDatabaseProductVersion()***
* ***getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)***

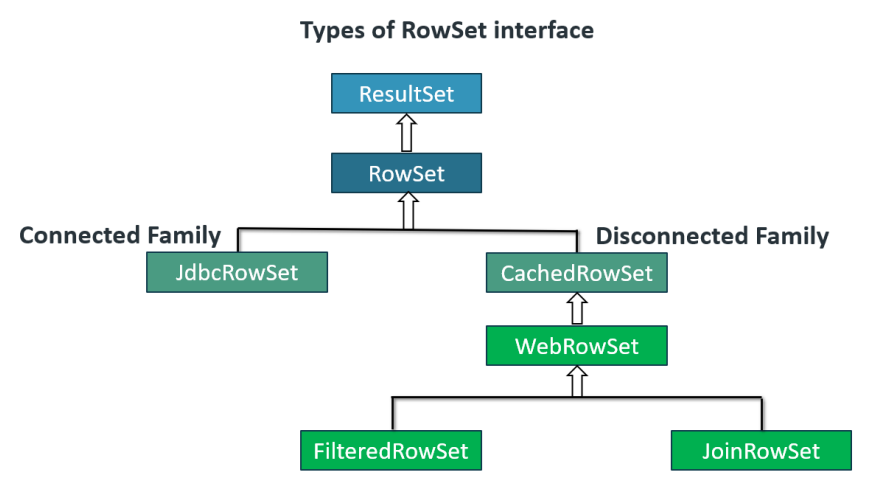
**Topic – 6: CallableStatement Interface**

**Introduction**

* ***CallableStatement*** is extension of ***PreparedStatement***.
* Has methods for executing stored procedures in a database.
* **Stored procedure:** Pre-compiled SQL statements within a database.

**RowSet Interface**

* **JDBC RowSet** provides facility to display data in form of table in console.
* It is wrapper of **ResultSet**.



**JdbcRowSet**

* Has scrollable and updatable abilities.

**CachedDataSet**

* Can operate without connecting to its data source.
* Also known as disconnected **RowSet** object.
* **Caches** data to its own memory instead of retrieving from the DB.

**WebRowSet**

* An extension to **CachedRowSet** with an additional capability.
* This additional capability is that it can read & **write XML** document.
* It can write itself to an **XML** document too.

**FilteredRowSet**

* It is a further extension to **WebRowSet**.
* Filters data as per criteria, done using **WHERE** clause.

**Topic – 7: Introduction To Maven**

**Introduction**

* Automation & management tool.
* Developed by ***Apache Software Foundations***.
* Assists developers with managing Java projects.
* Allows creating projects, dependency & documentation using **Project Object Model** & **Plugins**.

**Key Features**

* Keeps **default values** to reduce configuration time.
* Contains project templates.
* Loads dependencies into the project folder.
* Compiler plug-in support.

**User Guide**

* Maven can be configured using **project object model (POM)** which is stored in **pom.xml-file**.
* Plugins can be simply modified using ***<plugins>*** tag in **XML** file.

**Build Lifecycle**

|  |  |
| --- | --- |
| **Phase** | **Description** |
| **Prepare resources** | **Allows customizing resource copying.** |
| **Validate** | **Project configuration is validated.** |
| **Compile** | **Compiles source code into bytecode.** |
| **Test** | **Tests the project.** |
| **Packaging** | **Packages compiled code into JAR, WAR etc (*artifacts*).** |
| **Installation** | **Installs artifact into local repository.** |
| **Deployment** | **Copies artifacts to remote repository.** |

**Maven Repository**

* Maven repositories contain **JAR files** with some **metadata**.
* These **metadata** are **POM files** of projects with **external dependencies**.
* This **metadata** helps download the required dependencies.

**Types Of Maven Repositories**

* **Local repository:** It is the **first place** where dependencies are searched. Present in user’s machine itself with all required dependencies for current projects.
* **Central repository:** It is the **second place** where dependencies are searched. When dependencies aren’t found in local repository, then they are searched here & downloaded into **local repository**.
* **Remote repository:** Repository on web from where **Maven** dependencies can be downloaded. These can be downloaded into **local repository** if required.

**Uses Of Maven**

* ***Artifacts*** can be easily added into projects **without** separate scripting.
* Contains various **documents** about the resources being used in a project.
* Eases **compiling**, **testing**, **packaging** & **deploying** process.
* Keeps directory structure **simple** to understand.
* Plugins provide code analysis.
* Highly **customizable** for the developers.