# Day 10

## Metadata

- Data about data or data which describe another data is called metadata.
- Metadata of Interface

1. What is name of the interface.
2. In which package it is declare.
3. Which is super interface of the interface
4. Which is its access modifier of it
5. Which annotations has been used on interface
6. Which members are declared inside interface.

- Metadata of class

1. What is name of the class.
2. In which package it is declared
3. Which access modifier of the class
4. Which is super class of the class
5. Which interfaces it has implemented
6. Which annotations has been used on class
7. Whether class is final or abstract
8. Which are the members of class

- Metadata of field

1. What is name of the field
2. Which is access modifier of the field
3. Whether field is static, final, transient.
4. Which is type of field
5. Which annotations has been used on field.
6. Whether field is inherited or declared only.
7. Which is default value of field

- Metadata of method

1. What is name of method
2. Which is access modifier of the method
3. Whether method is abstract, final, static or synchronized
4. Which return type of the method
5. What is the parameter information
6. Which exceptions method throws
7. Which annotations has been used on method
8. Whether method is inherited, overriden or declared only method

## Application of metadata

1. Java compiler convert .java file into .class file. This file contains bytecode and metadata. Due to this metadata, there is no need to include header file in java.
2. Intelisence is a IDE feature which helps developer to write code. To display Type information inside intelisence windows IDE implicitly use metadata.
3. Metadata helps JVM to create clone of instance or to serialize state of java instance.
4. To verify bytecode, verfier implicitly use metadata.
5. To keep track of life time of object, garbage collector implictly use metadata.

## Reflection

- It is a java language feature which provides types( interface + classes ) to explore and process metadata.
- If we want to use reflection then we should use types declared in
    1. java.lang
    2. java.lang.reflect
- Types declared in java.lang.reflect package
    1. Array
    2. Constructor
    3. Field
    4. Method
    5. Modifier
    6. Parameter
- Type(s) declared in java.lang package
    1. Class

## Application of reflection

1. To display type information( metadata ) on terminal javap tool implicitly use reflection.
2. To display type information( metadata ) in intelisence window IDE implicitly use reflection.
3. To access value of private field, debugger implicitly use reflection.
4. If we want to manage behavior of application at runtime then we should use reflection.
5. If we want to map object/instance to database-Table record then we should use reflection

## java.lang.Class

- It is a final class declared in java.lang package.
- Instances of the class java.lang.Class represent classes and interfaces in a running Java application.
- Class has no public constructor. Instead Class objects are constructed automatically by the Java Virtual Machine.
- In simple words, instance of java.lang.Class represent metadata of loaded type.
- To use reflection, it is necessary to get reference of Class class instance associated with loaded type.
- Class loading mechanism of java is dynamic.

## Method/proces to get reference of java.lang.Class instance.

1. Using getClass() method.

```
Integer n1 = new Integer(0);
Class<?> c = n1.getClass();
```

- It is native and final method of java.lang.Object class.
- If we want to get reference of Class class instance associated with concrete class then we should use "getClass()" method

2. Using ".class" syntax

```
Class<?> c = ResultSet.class;
```

- if we want to get reference of Class class instance associated with abstract class or interface then we should use .class syntax.

3. Using Class.forName() method

```
try( Scanner sc = new Scanner(System.in))
{
    System.out.print("F.Q. Class Name : ");
    String className = sc.nextLine();
    Class<?> c = Class.forName(className);
}
```

- If we want to get reference of Class class instance associated with F.Q.Type Name specified at runtime then we should use forName() method.
- invoke() is a method of Method class.
- Syntax: "Object invoke(Object obj, Object... args)" Object obj : reference of instance Object... args : the arguments used for the method call

## Java DataBase Connectivity( JDBC )

```
mysql -u sandeep -psandeep (press enter key)
SHOW DATABASES;
```

```
GRANT ALL ON *.* TO sandeep@localhost;
flush privileges;
```

```
CREATE DATABASE kdac_db;
USE kdac_db;
SELECT database();
```

```
CREATE TABLE employees
(
    emp_id INT PRIMARY KEY,
    emp_name VARCHAR( 50 ),
    salary FLOAT,
    join_date DATE
);
```

```
INSERT INTO employees
VALUES
( 1,'Nitin',85000,'1998-11-25'),
( 2,'Prashant',75000,'2000-04-12'),
( 3,'Sandeep',25000,'2006-12-26');
```

**Configuration Information**

- JDBC Version : 4.2
- DataBase Server : MySQL
- Database Version : 8.0.16
- Machine Name : localhost
- Schema / Database : kdac_db
- User Name : sandeep
- password : sandeep
- URL : "jdbc:mysql://localhost:3306/kdac_db"
- JDBC Driver : com.mysql.cj.jdbc.Driver
- JDBC Connector : mysql-connector-java-8.0.16.jar

**JDBC**

- Specification : { interfaces + abstract classes }
- JDBC is a specification defined by SUN/ORACLE
- Implementating JDBC specification is a job of Database vendor
- To access data from relational database management system into java application, application developer must use JDBC.
- If we want to reduce database vendor dependancy in the code then we should use JDBC.
- If want to use JDBC then we must import java.sql package.
- Interfaces declared in java.sql package:
    1. Driver
    2. Connection
    3. Statement
    4. PreparedStatement
    5. CallableStatement
    6. ResultSet
    7. Blob
    8. Clob

    9. NClob
    10. DatabaseMetaData
    11. ResultSetMetaData
    12. ParameterMetaData
- Classes declared in java.sql package:
    1. DriverManager
    2. Date
    3. Time
    4. Timestamp
    5. Types
- SQL is a language of database and java is a language of java application.
- Driver is a program which is responsible for converting java request into sql request and sql response into java response.

### Types of JDBC Driver

1. TYPE - I
2. TYPE - II
3. TYPE - III
4. TYPE - IV

#### Type-I Driver

- It is also called JDBC-ODBC Bridge Driver.
- e.g : sun.jdbc.odbc.JdbcOdbcDriver
- Open DataBase Connectivity ( ODBC ) is a specification defined by Microsoft to connect any application to the database.
- Job of Type-I Driver is to convert JDBC calls into ODBC Calls or vice versa.
- Job of ODBC Driver is to convert ODBC calls into DB specific Calls or vice versa.
- Advantages:
    1. Easily available because it comes with JDK
    2. Easy to use/install( DSN configuration is required )
    3. It is database independant driver.
- DisAdvantages:
    1. It is platform dependant driver.
    2. Since multiple layers are involved slower in performance
    3. It is obsolete driver. Dont come with JDK 1.8 and onwards.
- Type-I driver is database independant but platform dependant Driver.

#### Type-II Driver

- It is also called as Native API Driver.
- e.g : Oracle Call Interface ( OCI ) Driver.
- It is platform dependant as well as database dependant driver.
- Java Native Interface( JNI ) is a java language feature that is used to access native code(C/C++) into java.
- Advantages
    1. It is not depend on ODBC driver hence faster than Type-I driver.

    2. We can use it on Windows as well Linux platform
- DisAdvantages
    1. Not all the vendors provide native API libraries
    2. It is database dependant driver( Type-II driver of windows and Linux is different ).

#### Type-III Driver

- It is also called as network protocol driver.
- e.g : RMI Web Logic Driver
- It is pure java driver which require seperate network infrastructure.
- It follows n-tier architecture.
- It is platform independant as well as database independant driver
- Advantages:
    - It is portable driver
    - No need of ODBC driver and vendor provided native API library
    - We can use it to fetch data from multiple different databases.
- DisAdvantages:
    - Expensive to use.

#### Type-IV Driver

- It is also called as Database protocol driver / thin driver
- It is pure java driver.
- It is database dependant but platform independant driver.
- e.g : com.mysql.cj.jdbc.Driver
- Advantages
    1. It implicitly use socket programming hence no need to use odbc driver and vendor provided native API libraries.
    2. Since implemented in java, it is portable.
    3. Easy to install and use.
    4. It can be used with different type of java application :
        1. CUI
        2. GUI
        3. Web Application
- DisAdvantages
    1. It is database dependant Driver.

### JDBC Versions

Since 1.8 --- JDBC 4.2 Since 1.7 --- JDBC 4.1 Since 1.6 --- JDBC 4.0 Since 1.4 --- JDBC 3.0 Since 1.2 --- JDBC 2.0 Since 1.1 --- JDBC 1.0

### Steps to connect Java Application to Database

Step 0 : Add JDBC connector( .jar ) in buildpath / classpath / runtime classpath. Step 1 : Load and register Driver ( Since 1.8 it is optional ). Step 2 : Establish connection using users credential. Step 3 : Create

Statement/PreparedStatement/CallableStatement to execute query. Step 4 : Prepare and Execute Query Step 5 : Close resources.