Object Oriented Programming

ICT2122

Database Connectivity

P.H.P. Nuwan Laksiri
Department of ICT
Faculty of Technology
University of Ruhuna

Lesson 07

Recap – File Handling

- Getting to Know About Streams
- *Streams*
  - *Character Streams*
  - *Binary Streams*
- *Character Streams*
  - *Reading*
  - *Writing*
- *Binary Streams*
  - *Reading*
  - *Writing*

Outline

- What is an API
- JDBC Introduction
- Why JDBC
- JDBC Components
- JDBC Architecture
- JDBC Drivers
- Basic steps to use a database in Java
- Handling Exceptions

API – What is it?

- API stands for application programming interface.
- Java SE API
  - The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing.
  - These APIs are in modules whose names start with ***java***.
  - https://docs.oracle.com/en/java/javase/19/docs/api/index.html

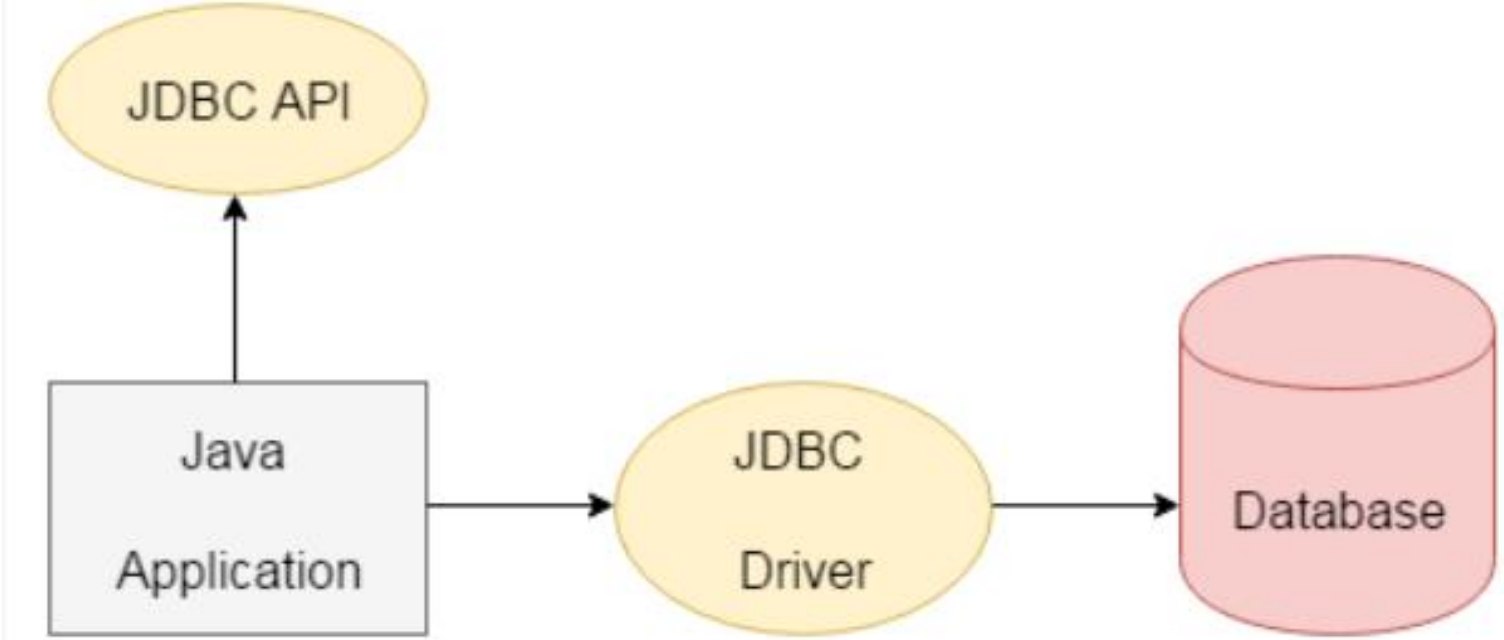
- On the other hand, API is a document that contains description of all the features of a product or software.
- It represents classes and interfaces that software programs can follow to communicate with each other.
- An API can be created for applications, libraries, operating systems, etc.

API – What is it?

- The Java language itself is very simple,
  - but Java comes with a library of classes that provide commonly used utility functions that most Java programs can't do without.
- This class library, called the Java API is as much a part of Java as the language itself.
- The real challenge of finding out how to use Java isn't mastering the language; it's mastering the API.
- The Java language has only about 50 keywords, but the Java API has several thousand classes, with tens of thousands of methods that you can use in your programs.

JDBC Introduction

- JDBC stands for *Java Database Connectivity*, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.
- JDBC is to connect and execute query with the database.
- JDBC API uses jdbc drivers to connect with the database.



JDBC Introduction

- JDBC allows to,
  - Establishing a connection with a database or other tabular data source
  - Sending SQL commands to the database
  - Processing the results

Why JDBC

- Before JDBC, ODBC API was the database API to connect and execute query with the database.
- But ODBC API uses ODBC driver which is written in C language (it's platform dependent and unsecured).
- That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

Product components of JDBC

- The JDBC API.
  - JDBC is a Java Database Connectivity API that lets you access virtually any tabular data source from a Java application.
- The JDBC Driver Manager.
  - Driver Manager is the backbone of the JDBC architecture. Defines objects which connect Java applications to a JDBC driver.
- The JDBC Test Suite.
  - Ensure that the JDBC drivers will run user's program or not .
- The JDBC-ODBC Bridge.
  - Database driver that utilize the ODBC driver to connect the database by translating JDBC method calls into ODBC function calls.

JDBC Architecture

- JDBC architecture consists of two layers,
  - JDBC API:

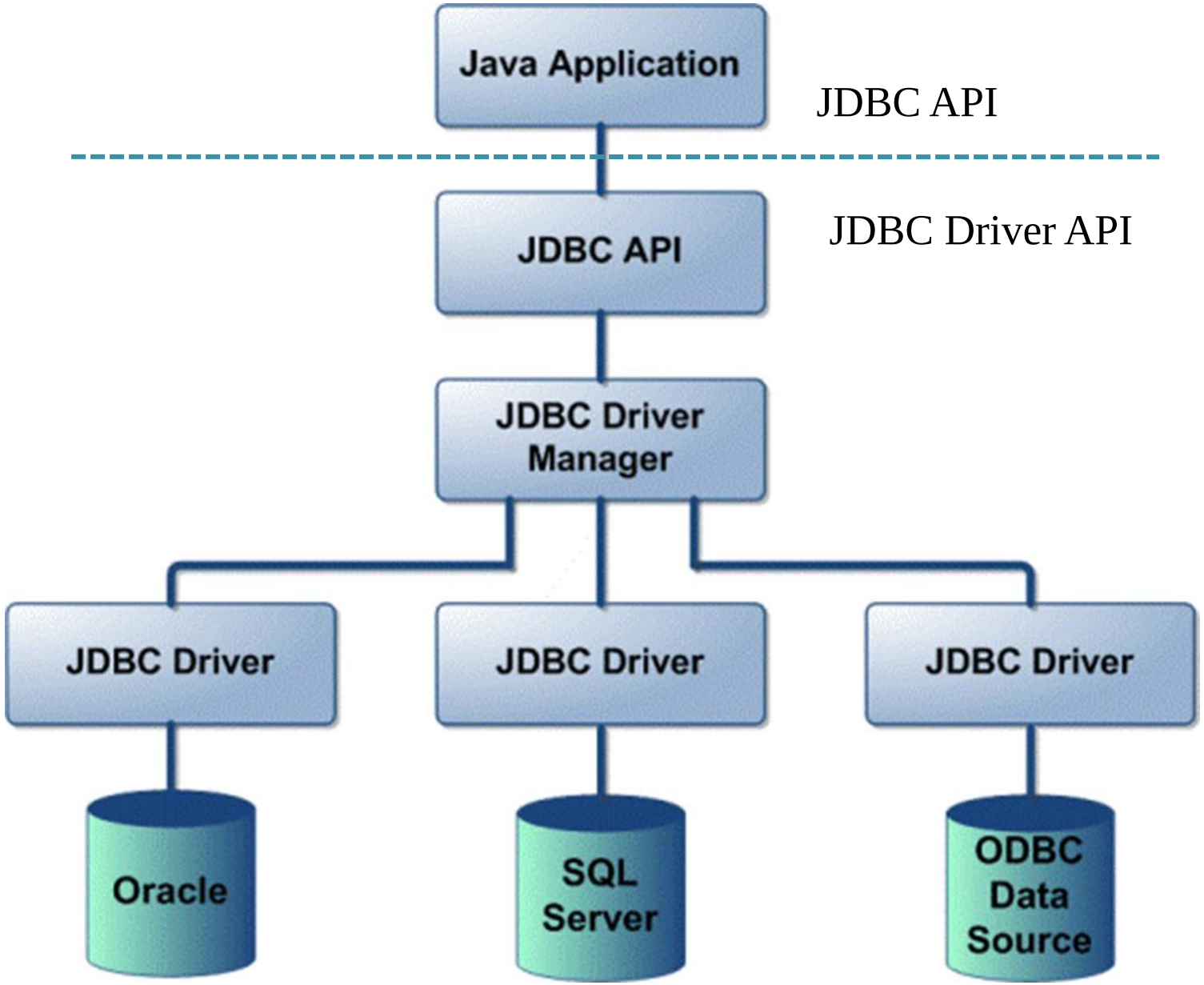
    This provides the application-to-JDBC Manager connection.
  - JDBC Driver API:

    This supports the JDBC Manager-to-Driver Connection.
- The JDBC API uses
  - a driver manager and
  - database-specific drivers

  to provide transparent connectivity to heterogeneous databases.

- The JDBC driver manager ensures that the correct driver is used to access each data source.
- The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.

JDBC Architecture



Common JDBC components

- The JDBC API provides the following interfaces and classes,

- **Driver manager**
  - This class manages a list of database drivers.
  - Matches connection requests from the java application with the proper database driver using communication sub protocol.
  - The first driver that recognizes a certain sub protocol under JDBC will be used to establish a database Connection.

- **Driver**
  - This interface handles the communications with the database server.

Common JDBC components

- **Connection**
  - This interface with all methods for contacting a database.
  - The connection object represents communication context, i.e., all communication with database is through connection object only.

- **Statement**
  - User use objects created from this interface to submit the SQL statements to the database.
  - Some derived interfaces accept parameters in addition to executing stored procedures.

Common JDBC components

- **ResultSet**
  - These objects hold data retrieved from a database after you execute an SQL query using Statement objects.

- **SQLException**
  - This class handles any errors that occur in a database application.

JDBC Driver

- JDBC Driver is a software component that enables java application to interact with the database.
- There are 4 types of JDBC drivers.
  - JDBC-ODBC bridge driver
  - Native-API driver

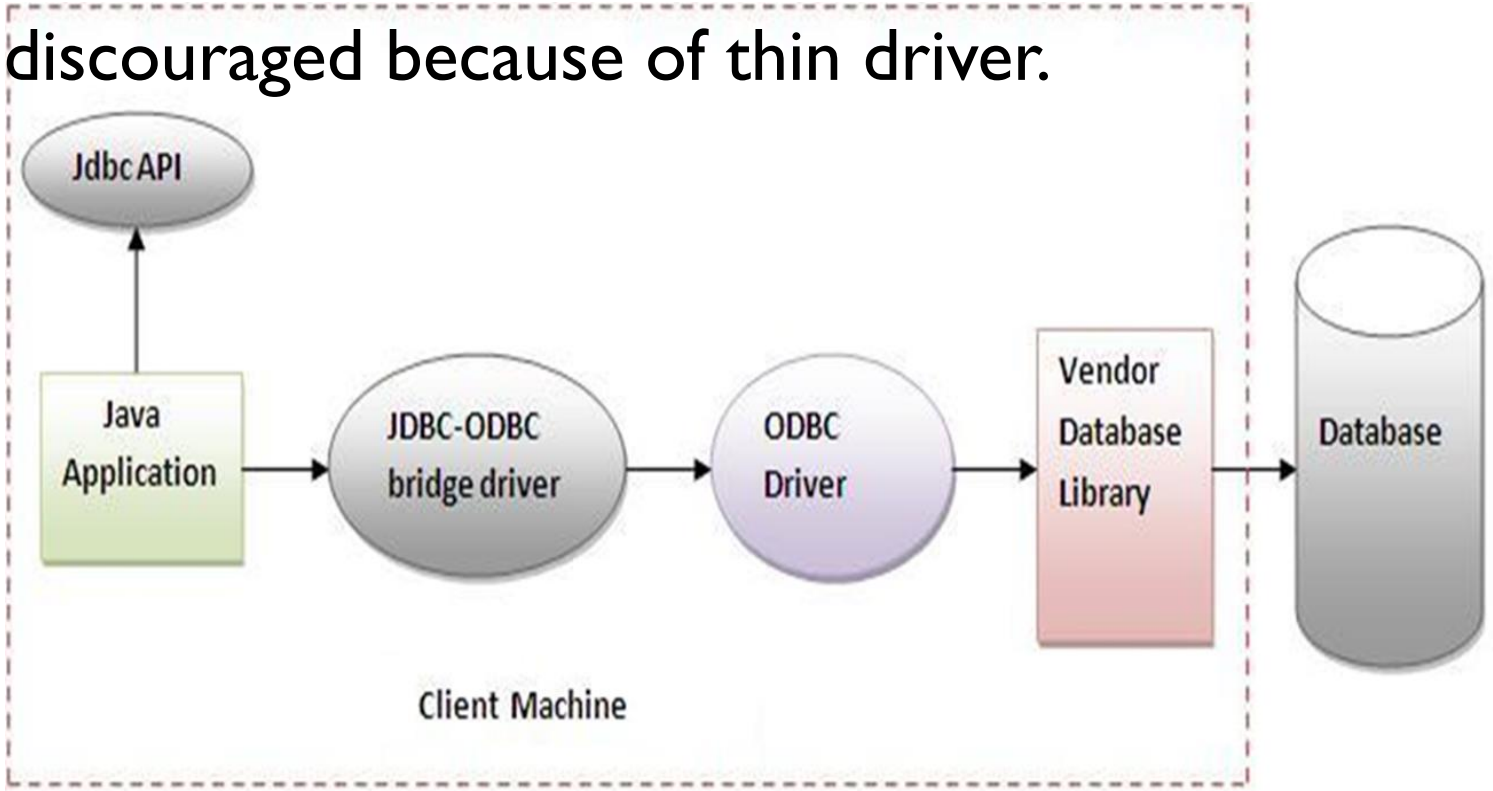
    (partially java driver)
  - Network Protocol driver

    (fully java driver)
  - Thin driver

    (fully java driver)

JDBC Driver - JDBC-ODBC bridge driver

- The JDBC-ODBC bridge driver uses ODBC driver to connect to the database.
- The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls.
- This is now discouraged because of thin driver.



JDBC Driver - JDBC-ODBC bridge driver

Advantages

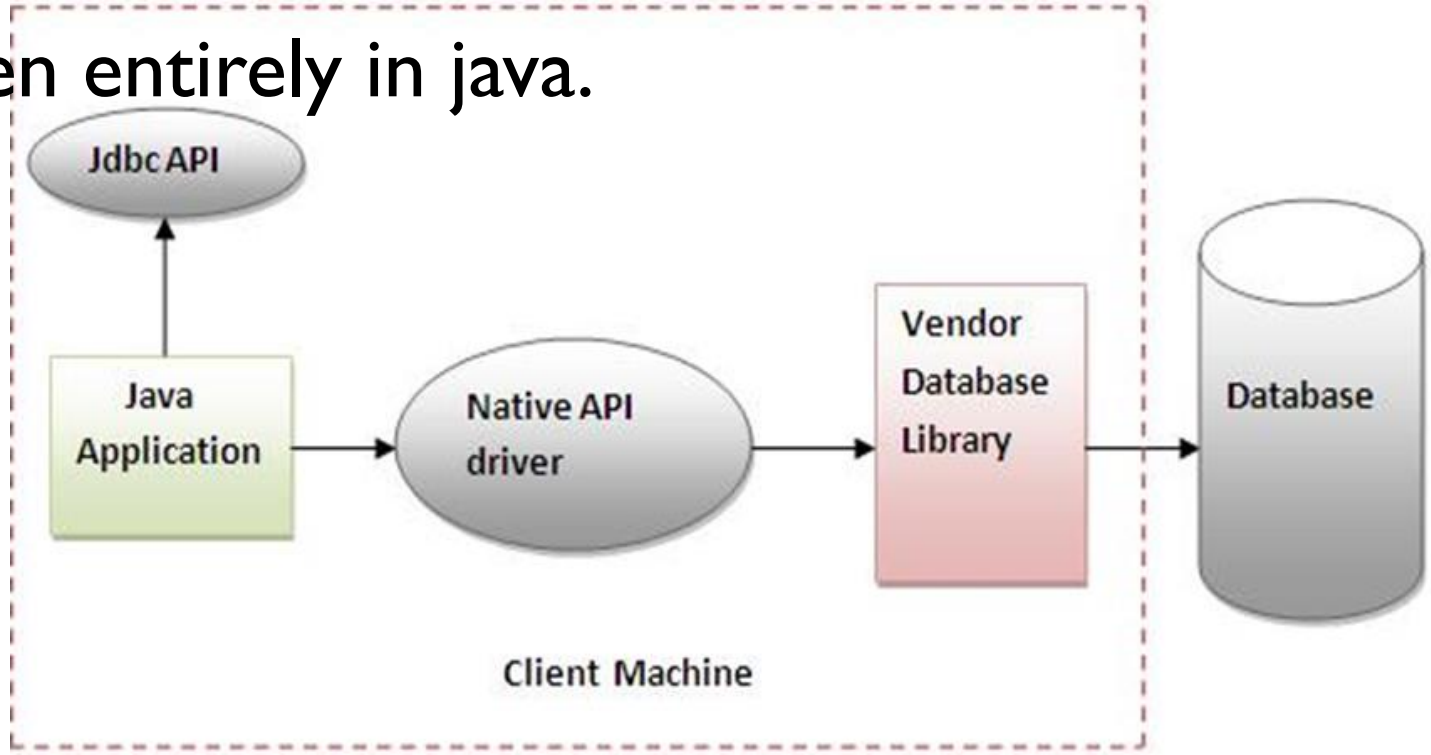
- easy to use.
- can be easily connected to any database.

Disadvantages

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

JDBC Driver – Native API driver

- The Native API driver uses the client-side libraries of the database.
- The driver converts JDBC method calls into native calls of the database API.
- It is not written entirely in java.



JDBC Driver – Native API driver

Advantages

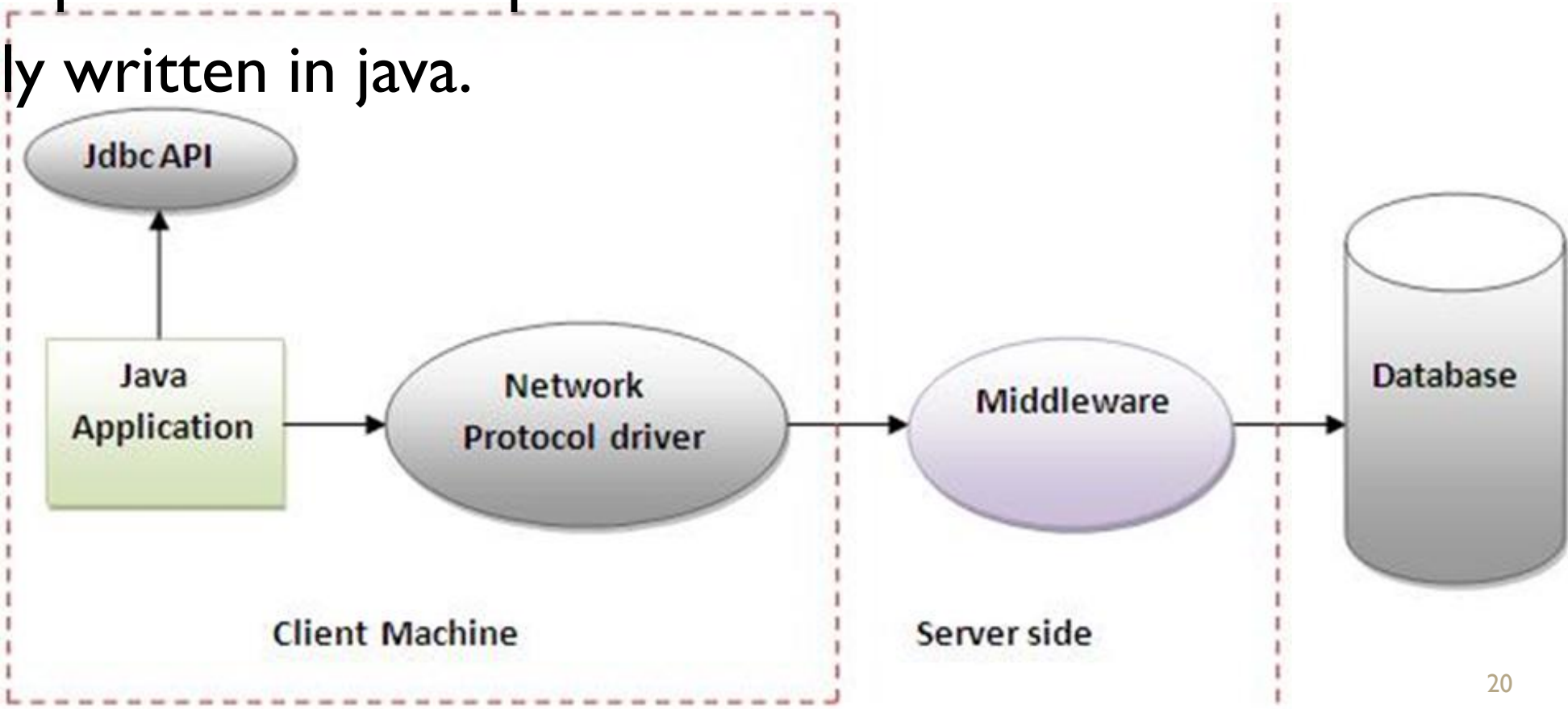
- performance upgraded than JDBC-ODBC bridge driver.

Disadvantages

- The Native driver needs to be installed on each client machine.
- The Vendor client library needs to be installed on client machine.

JDBC Driver – Network Protocol driver

- The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol.
- It is fully written in java.



JDBC Driver – Network Protocol driver

Advantages

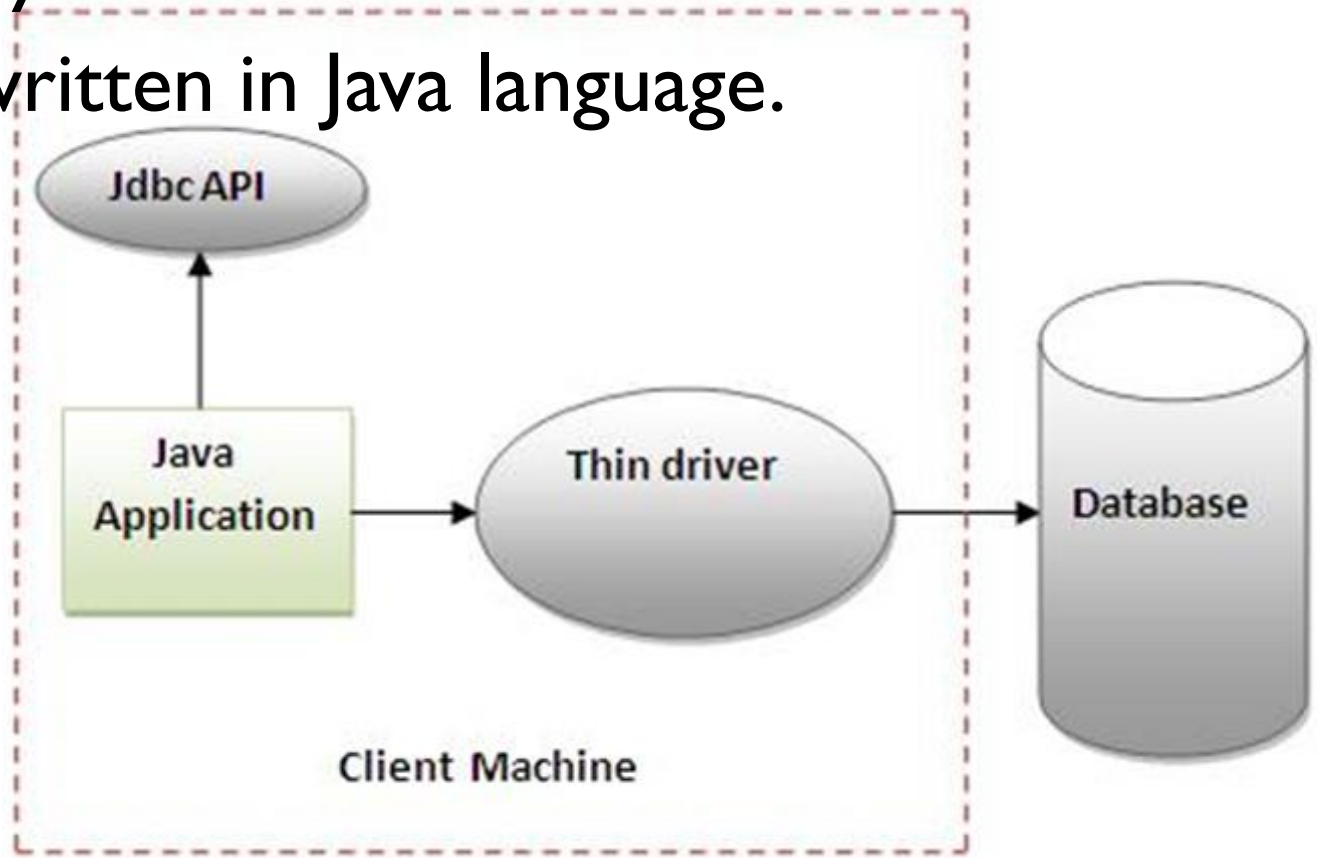
- No client-side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

JDBC Driver – Thin driver

- The thin driver converts JDBC calls directly into the vendor-specific database protocol.
- That is why it is known as thin driver.
- It is fully written in Java language.



JDBC Driver – Thin driver

Advantages

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantages

- Drivers depend on the Database.

MySQL Connector/J

- MySQL provides connectivity for client applications developed in the Java programming language with MySQL Connector/J.
- Connector/J implements the Java Database Connectivity (JDBC) API, as well as a number of value-adding extensions of it.
- MySQL Connector/J is a JDBC Type 4 driver, implementing the JDBC specification.

  (The Type 4 designation means that the driver is a pure Java implementation of the MySQL protocol and does not rely on the MySQL client libraries.)

Basic Steps to Use a Database in Java

- There are 05 basic steps to connect any java application with the database in java using JDBC,
  - **Register the driver class**
  - **Creating connection**
  - **Creating statement**
  - **Executing queries**
  - **Closing connection**

Register the Driver Class

- The forName() method of Class is used to register the driver class.
  - This method is used to dynamically load the driver class.
- The most common approach to register a driver is to use Java's Class.forName() method, to dynamically load the driver's class file into memory, which automatically registers it.

Syntax

Class.forName(“com.mysql.cj.jdbc.driver");

Or

Class.forName(" com.mysql.cj.jdbc.driver ").newInstance();

Creating Connection

- The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax

Connection con = DriverManager.getConnection(URL,UserName,Password);

- *URL* is an address that points to your database.
- *UserName* is the user of the database.
- *Password* is the password for the database.

Creating Statement

- The createStatement() method of Connection interface is used to create statement.
- The object of statement is responsible to execute queries with the database

Syntax

Statement stmt = con.createStatement();

- We use Statement objects in order to
  - Query the database
  - Update the database

Creating Statement

- There are three different kinds of statements

- Statement
  - Used to implement simple SQL statements with no parameters.
- PreparedStatement - Extends Statement
  - Used for precompiling SQL statements that might contain input parameters.
- CallableStatement - Extends PreparedStatement
  - Used to execute stored procedures that may contain both input and output parameters.

Executing queries

- The executeQuery() method of Statement interface is used to execute queries to the database.
- This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax

ResultSet rs = stmt.executeQuery(query);

- "Query" is a SQL query
  - Ex : SELECT *
    FROM students

Executing queries

Following methods are used in Statement interface to Execute SQL Statements

- **execute**
  - Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false

    boolean execute (String sql) throws SQLException

- **executeUpdate**
  - Used for data manipulation: insert, delete, update, create table, etc.
  - Returns the number of rows modified

    int executeUpdate(String sql) throws SQLException

- **executeQuery**
  - Used for sending queries(select queries) ,returns a ResultSet object representing the query result.

    ResultSet executeQuery(String sql) throws SQLException

Closing Connection

- By closing connection object statement and ResultSet will be closed automatically.
- The close() method of Connection interface is used to close the connection.

Syntax

con.close();

- con is the connection object created at the beginning.

Connect to mysql Database- Example

Following information are needed for the mysql database:

- Driver class
  - The driver class for the mysql database is com.mysql.cj.jdbc.Driver.
- Connection URL
  - The connection URL for the mysql database is jdbc:mysql://localhost:3306/tecruh
    - jdbc is the API
    - mysql is the database
    - localhost is the server's name on which mysql is running, we may also use IP address,
    - 3306 is the port number and tecruh is the database name.
  - Username
    - The default username for the mysql database is root.
  - Password
    - Password is given by the user at the time of installing the mysql database.
    - In this example, we are going to use 123 as the password.

Connect to mysql Database - Example

- Create "tecruh" database and "student" table.

```
CREATE DATABASE tecruh;

CREATE TABLE student(
        id int(10) PRIMARY KEY,
        name varchar(40),
        age int(3));
```

Connect to mysql Database - Example

```
// 1. importing the mysql library
import java.sql.*;  //importing the mysql library
class MysqlCon{
    public static void main(String args[]){
        try{
            // 2. register the driver
          Class.forName("com.mysql.jdbc.Driver");


            // 3. establish the connection
         Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/tecruh","root",“123");
          //here tecruh is database name, root is username and 123 is password


            // 4. create statement
         Statement stmt=con.createStatement();
```

Connect to mysql Database - Example

```
        // 5. execute the query to retrieve data from table
        ResultSet rs=stmt.executeQuery("select * from student");


        // 6. traverse through the result set
        while(rs.next())
                System.out.println(rs.getInt(1)+""+rs.getString(2)+""+rs.getString(3));


        // 7. close the connection
        con.close();
      }
      catch(Exception e){
        System.out.println(e.getMessage());
    }
  }
}
```

Prepared Statements

- Prepared Statements are used for queries that are executed many times
- They are parsed (compiled) by the DBMS only once
- Column values can be set after compilation
- Prepared statement is used to execute parameterized query.
- Instead of values, use '?'
- Hence, Prepared Statements can be thought of as statements that contain placeholders to be substituted later with actual values

Example of parameterized query:

```
String sql="insert into emp values(?,?,?)";
```

Why use Prepared Statement?

- Improves performance
  - The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

Example of prepared statement

```
String val = "abc";
PreparedStatement pstmt = con.prepareStatement("select * from R where A=?");
pstmt.setString(1, val);
ResultSet rs =  pstmt.executeQuery();
```

Get ResultSet

- ResultSet objects provide access to the tables generated as results of executing a Statement queries
- The table rows are retrieved in sequence
- A ResultSet maintains a cursor pointing to its current row .
- The next() method moves the cursor to the next row.

ResultSet Methods

- Type getType(int columnIndex)
  - returns the given field as the given type
  - indices start at 1 and not 0!
- Type getType(String columnName)
  - same, but uses name of field
  - less efficient
- For example:

  getString(columnIndex),
  getInt(columnName),
  getTime(columnName),
  getBoolean(columnName),

Handling Errors with Exceptions

- If a statement in the try block throws an exception or warning, it can be caught in one of the corresponding catch statements.
- Reason is programs should recover and leave the database in a consistent state after any error.
- E.g., you could rollback your transaction in a catch { …} block or close database connection and free database related resources in finally {…} block

HandsOn

- Refer “Practical 09”

Summary

- What is an API
- JDBC Introduction
- Why JDBC
- JDBC Components
- JDBC Architecture
- JDBC Drivers
- Basic steps to use a database in Java
- Handling Exceptions

References

- https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html
- https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-api-changes.html

- How To Program (Early Objects)
  - By H .Deitel and P. Deitel
- Headfirst Java
  - By Kathy Sierra and Bert Bates

Questions ???

Thank You