Core Java 8 and Development Tools

Lesson 16 : Java Database Connectivity (JDBC 4.0)

Capgemini

Lesson Objectives

After completing this lesson, participants will be able to

- Understand concept database connectivity architecture
- Work with JDBC API 4.0
- Access database through Java programs
- Understand advance features of JDBC API

16.1: Java Database Connectivity
JDBC – an introduction

Java Database Connectivity (JDBC) is a standard SQL database access interface, providing uniform access to a wide range of relational databases.

JDBC allows us to construct SQL statements and embed them inside Java API calls.

JDBC provides different set of APIs to perform operations related to database; allows us to:

- Establish a connection with a database.
- Send SQL statements.
- Process the results

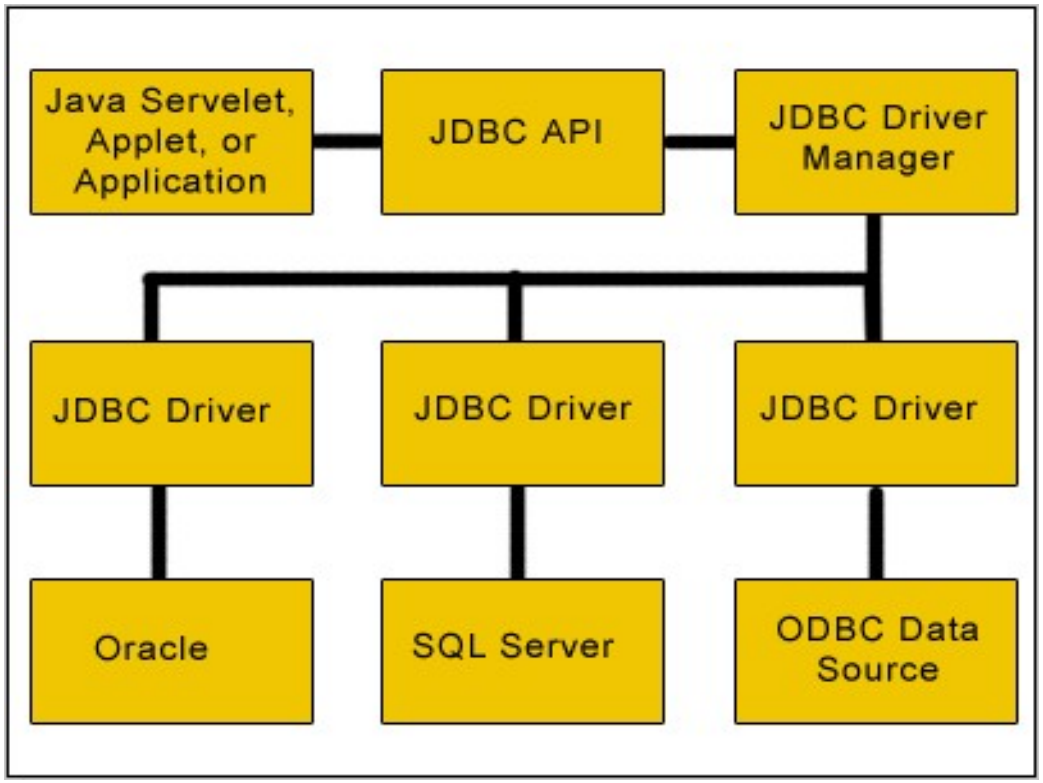
16.1: Java Database Connectivity
JDBC Features

JDBC exhibits the following features:

- Java is a write once, run anywhere language.
- Java based clients are thin clients.
- It is suited for network centric models.
- It provides a clean, simple, uniform vendor independent interface.
- JDBC supports all the advanced features of latest SQL version
- JDBC API provides a rich set of methods.

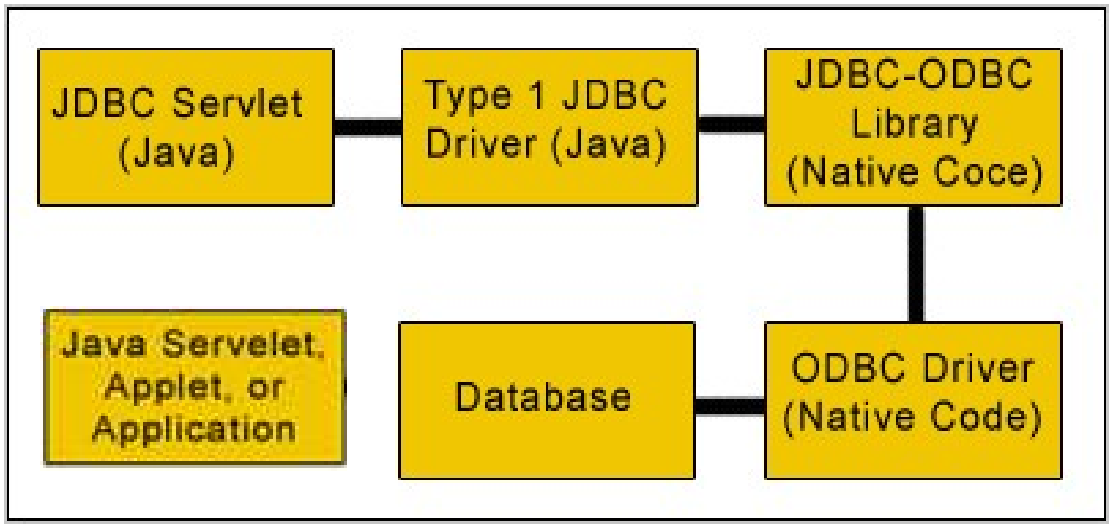
16.2: Database Connectivity Architecture
JDBC Architecture

Layers of JDBC Architecture



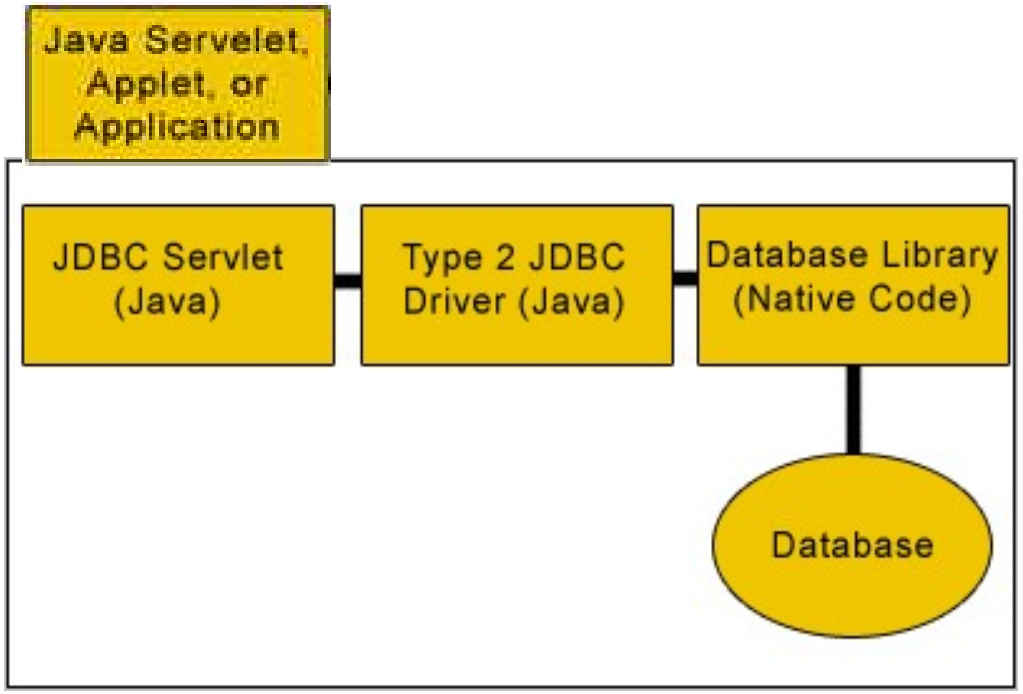
16.2: Database Connectivity Architecture
JDBC Driver

Type 1 – JDBC-ODBC Bridge



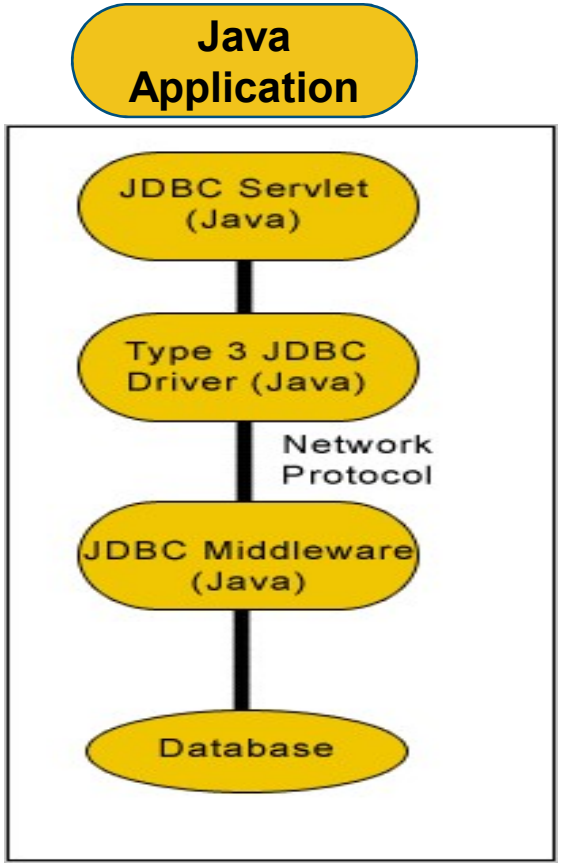
16.2: Database Connectivity Architecture
JDBC Driver

Type 2 – Java Native API



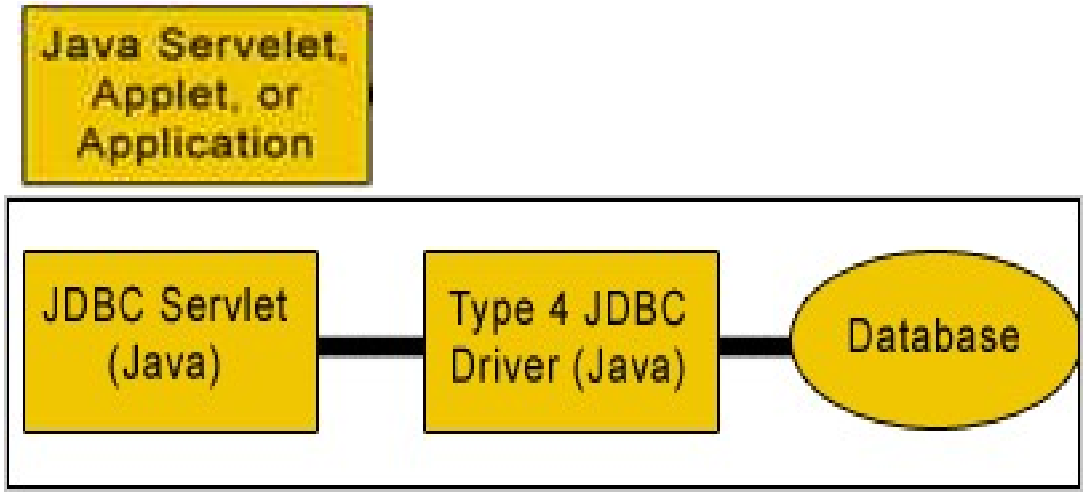
16.2: Database Connectivity Architecture
JDBC Driver

Type 3 – Java to Network Protocol



16.2: Database Connectivity Architecture
JDBC Driver

Type 4 – Java to Database Protocol

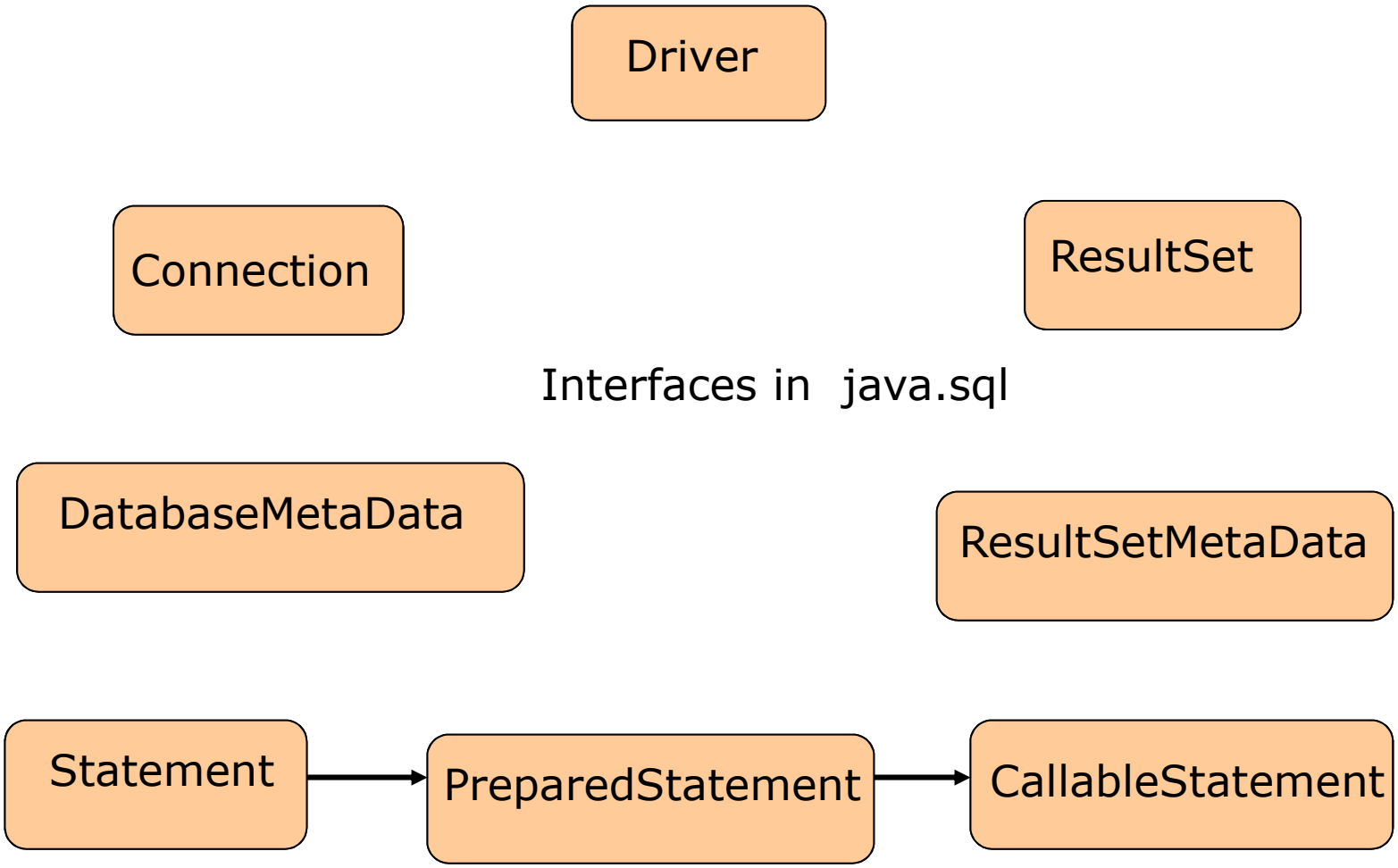


16.3: JDBC APIs
JDBC Packages

JDBC packages:

- java.sql.*
- javax.sql.*

16.3: JDBC APIs
JDBC API



16.4: Database Access Steps
JDBC Database Access

Database Access takes you through the following steps:

- Import the packages
- Register/load the driver
- Establish the connection
- Creating JDBC Statements
- Executing the query
- Closing resources

16.4: Database Access Steps
JDBC Database Access: Step-1

Step 1: Import the java.sql and javax.sql packages

- These packages provides the API for accessing and processing data stored in a data source.
- They include:
  - import java.sql.*;
  - import javax.sql.*;

16.4: Database Access Steps
JDBC Database Access: Step-2

Step 2: Register/load the driver

- Class.forName("oracle.jdbc.driver.OracleDriver");

  OR

- DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());

DriverManager class is used to load and register appropriate database specific driver.

The registerDriver() method is used to register driver with DriverManager

In JDBC 4.0, this step is not required, as when getConnection() method is called, the DriverManager will attempt to locate a suitable driver

16.4: Database Access Steps
JDBC Database Access: Step-3

Step 3: Establish the connection with the database using registered driver

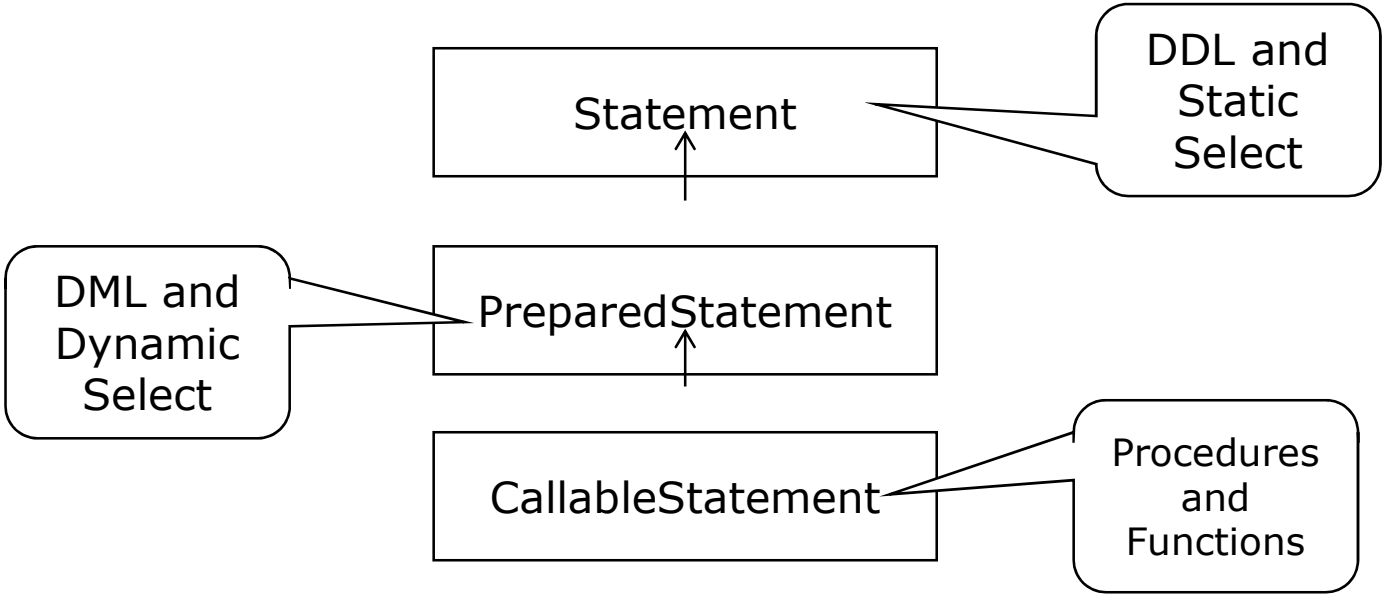
- String url = "jdbc:oracle:thin:@hostname:1521:database";
- Connection conn = DriverManager.getConnection (url, "scott", "tiger");

Once driver is loaded, Connection object is used to establish a connection with database.

16.4: Database Access Steps
JDBC Database Access: Step-4

Once connection is established with database, statement object can be used to execute different types of SQL queries

JDBC API support different statement interfaces depending upon type of query to be executed



16.4: Database Access Steps
JDBC Database Access: Step-4

Step 4: Create Statement

- Statement:
  - Statement st=conn.createStatement();
- PreparedStatement:
  - PreparedStatement pst=conn.prepareStatement("SELECT* FROM emp WHERE eno=?");
    pst.setInt(1, 100);
- CallableStatement:
  - CallableStatement cs=conn.prepareCall("{ call add() }") ;

Once statement object is created, use one of the following method to execute the query

Operation	Method
Select	ResultSet executeQuery(String query)
DML	int executeUpdate(String query)
DDL	boolean execute(String query)

16.4: Database Access Steps
JDBC Database Access: Step-5 (Querying Data)

Step 5: Retrieve data from table

- Statement:

```
Statement st=conn.createStatement();
ResultSet rs=st.executeQuery("SELECT * FROM emp");
while(rs.next()){
System.out.println("Emo No = "+rs.getInt("eno"));
System.out.println("Emo Name = "+rs.getString("ename"));
```

- Output:
  - Display the number and name of all employees

16.4: Database Access Steps
JDBC Database Access: Step-5 (inserting data)

Step 5: Insert data into table

- PreparedStatement:

```
String query = “INSERT INTO emp VALUES(?,?,?)”
PreparedStatement st=conn.prepareStatement(query);
st.setInt(1, 110);
st.setString(2, “xyz”);
st.setString(3, “Pune”);
int rec = st.executeUpdate();
System.out.println(rec + “ record is inserted”));
```

16.4: Database Access Steps
JDBC Database Access: Step-5 (updating data)

Step 5: Update table data

- Statement:

```
String query = "update emp set ecity=? where eno<?"
PreparedStatement st=conn.prepareStatement(query);
st.setString(1,"Mumbai");
st.setInt(2,1000);
int res = st.executeUpdate);
System.out.println(red + " records updated"));
```

16.4: Database Access Steps
JDBC Database Access: Step-5 (deleting data)

Step 5: Delete table data

- PreparedStatement:

```
String query = “delete from emp where eno<?”
PreparedStatement st=conn.prepareStatement(query);
st.setInt(2,1000);
int res = st.executeUpdate);
System.out.println(red + “ records deleted”));
```

16.4: Database Access Steps
JDBC Database Access: Step-6

Step 6: Closing resources

- Once done with data access following resources needs to be closed in order to free the underlying processes and release the memory
- resultSet.close();
- statement.close();
- connection.close();

16.4: Database Access Steps Demo

Execute the :

- Select.java
- Insert.java
- Delete.java
- PreparedStat.java programs

16.5: Calling database procedures
Calling Stored Procedures and Functions

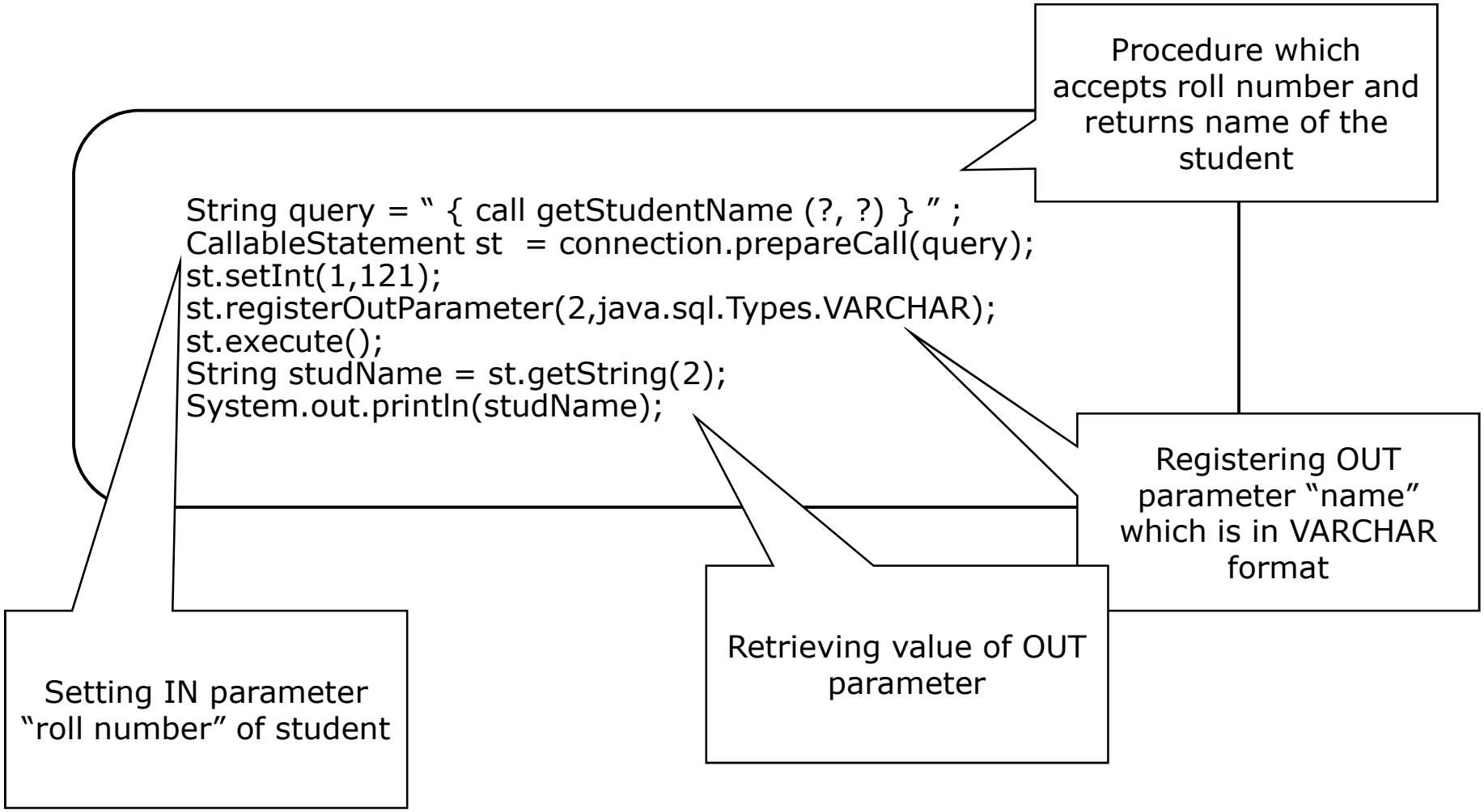
CallableStatement is used to invoke either procedure or function

CallableStatement interface extends PreparedStatement so as to support input and output parameters

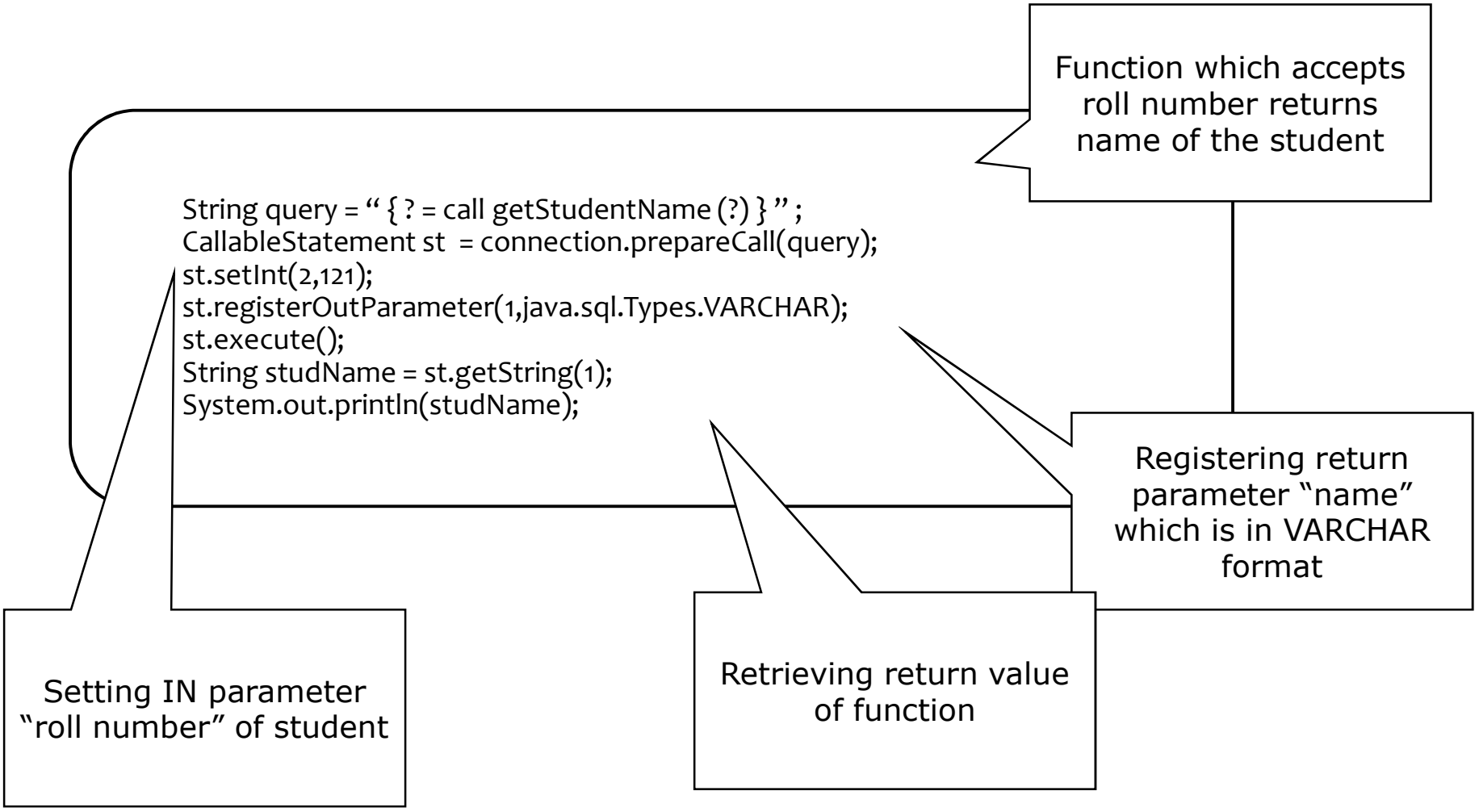
Steps to call procedure/function:

- Create callable statement object
- Call setXXX() method to set IN parameters
- Call registerOutParameter() method to register OUT parameters/function return value
- Call execute() to invoke the procedure/function
- Call getXXX() method to retrieve results from OUT parameters/function return value

16.5: Calling database procedures
Calling Stored Procedures



16.5: Calling database procedures
Calling Stored Functions



16.6: Using Transaction Transaction

A transaction is a set of one or more statements that are executed together as a unit, so either all of the statements are executed, or none of the statements is executed.

Java application uses Java Transaction API(JTA) to manage the transactions.

16.6: Using Transaction
Transaction (cntd…)

By default, a connection object is in auto-commit mode ie conn.setAutoCommit(true);

Disable the auto-commit mode to allow two or more statements to be grouped into a transaction:

Example: con.setAutoCommit(false);

```
try{
    con.setAutoCommit(false);
    sql-statement-1;
    sql-statement-1;
    ….
    conn.commit();
}
```

Transactional unit

commit the transaction

16.6: Using Transaction
Transaction (cntd…)

The rollback() method plays an important role in preserving data integrity in the transaction.

When you want to undo the changes of half done transaction due to SQLException:

```
try {
        conn.setAutoCommit(false);
          // perform transactions
        conn.commit();

    } catch (SQLException e) {
     conn .rollback() ;
     } finally {
         con.setAutoCommit(true);
     }
```

16.6: Using Transaction
Transaction (cntd…)

Savepoint is marking to roll back the transaction till the particular statement in the transaction.

You can set the multiple save points in the transaction.

Example of Setting Savepoint:

```
Savepoint svpt1 = conn.setSavepoint("SAVEPOINT_1");
……
con.rollback(svpt1);
```

Example of Releasing Savepoint:

```
conn.releaseSavepoint(svpt1);
```

Demo : Transaction

Execute the Transaction.java program

16.7: JDBC Best Practices
Best Practices

Some of the best practices in JDBC:

- Selection of Driver
- Close resources as soon as you're done with them
- Turn-Off Auto-Commit – group updates into a transaction
- Business identifiers as a String instead of number
- Do not perform database tasks in code
- Use JDBC's PreparedStatement instead of Statement when possible

16.7: JDBC Best Practices
Best Practices

16.8: Java Database Connectivity
Lab : JDBC

Lab 11: Property Files and JDBC 4.0

Summary

In this lesson, you have learnt:

- Establishing the connection with database to perform database operations
- Different types of statement creation
- Different ways of executing statements
- Transaction management using JDBC APIs
- Use of connection pooling to increase the performance of application
- And best practices for JDBC applications

Review Question

Question 1 : Which Statement is used when you want to pass parameter to the query?

- **Option 1 :** Statement
- **Option 2 :** PreparedStatement
- **Option 3 :** CallableStatement

Question 2 : _____ method is best suited to execute DDL Queries.

Question 3 : By default, a connection object is in auto-commit mode

- True/False