Lets Start With JDBC……

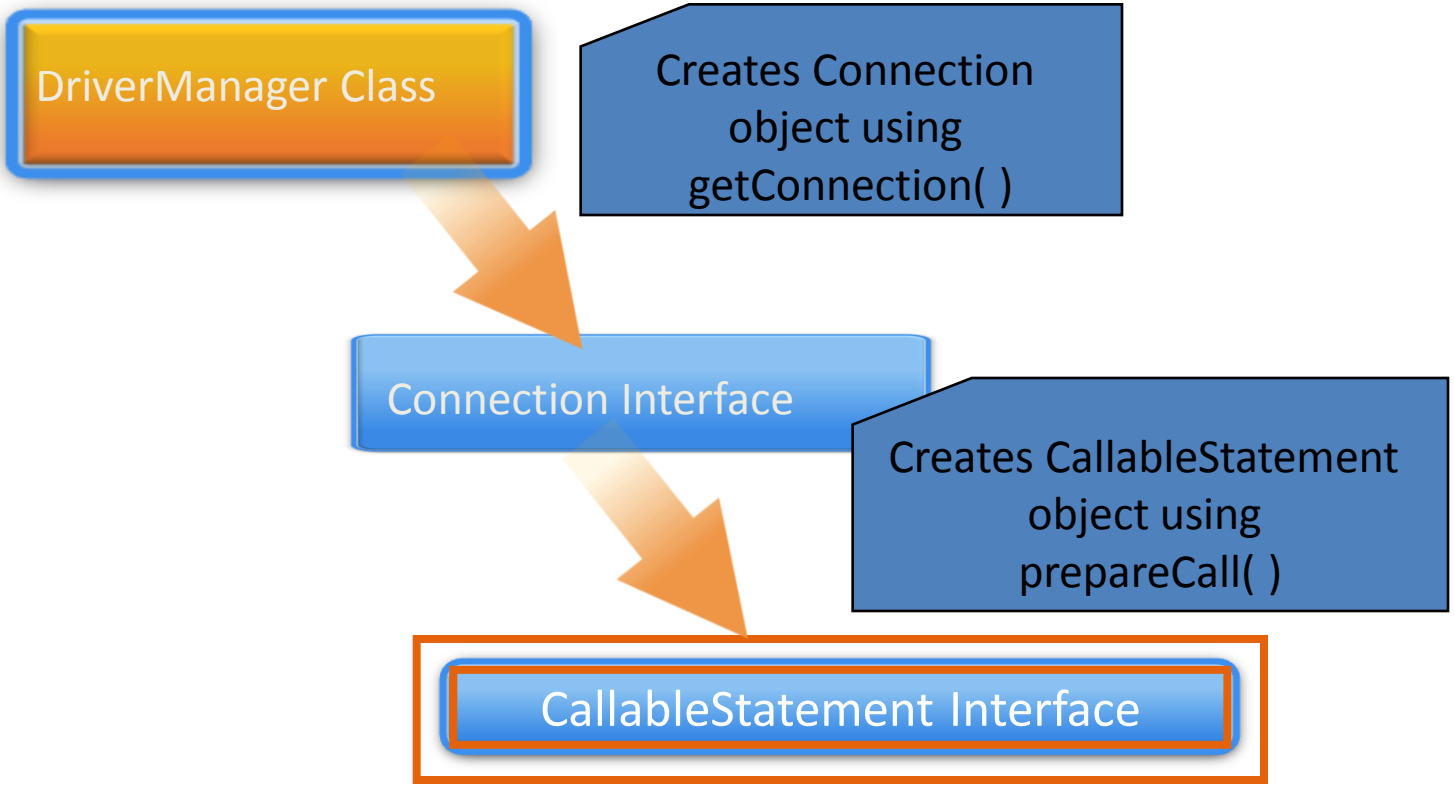
JDBC | Query To Database

Query database (Select or insert/update/delete)

- Create Statement Object
  - Create Object of Statement Interface OR
  - Create Object of PreparedStatement Interface OR
  - Create Object of CallableStatement Interface
- Execute the Query
  - Using Methods of CallableStatement Interface

JDBC | Query To Database using CallableStatement

Create object of CallableStatement Interface



JDBC | Steps for CallableStatement

The basic steps are:

Step 1: Connection.prepareCall method to create a CallableStatement object.

Step 2: CallableStatement.setXXX methods to pass values to the input (IN) parameters.

Step 3: The CallableStatement.registerOutParameter method to indicate which parameters are output-only (OUT) parameters or input and output (INOUT) parameters.

Step 4: One of the following methods to call the stored procedure:

CallableStatement.executeUpdate: Invoke this method if the stored procedure does not return result sets.

CallableStatement.executeQuery: Invoke this method if the stored procedure returns one result set.

CallableStatement.execute: Invoke this method if the stored procedure returns multiple result sets.

Step 5: If the stored procedure returns result sets, retrieve the result sets. Invoke the CallableStatement.getXXX methods to retrieve values from the OUT parameters or INOUT parameters.

Step 6: The CallableStatement.close method to close the CallableStatement object when you have finished using that object.

JDBC | Query To Database using CallableStatement

Create a procedure

Example :

```
Create or replace procedure remove (name varchar2) as
Begin
    Delete from emp where emp.Empname=name;
End;
```

JDBC | Query To Database using CallableStatement

Create a object using prepareCall Method of Connection Interface

Syntax :

Public CallableStatement prepareCall(String sql) throws SQLException

Example :

```
CallableStatement cst;
cst = con.prepareCall("{call remove( ? )}" );
```

JDBC | Query To Database using CallableStatement

Merge all values in SQL query where ? is given

To merge value of ? we have to use setXXX methods of CallableStatement. Syntax of setXXX methods:

setXXX(parameterIndex,parameterValue)

Example :

```
CallableStatement cst;
cst = con.prepareCall("{call remove( ? )}" );
cst.setString(1, "Dhruvi");
```

JDBC | Query To Database using CallableStatement

Execute Query using method of CallableStatement

Following are 3 different execute methods available in CallableStatement Interface:

1. executeQuery( ):
   public ResultSet executeQuery( ) throws SQLException
   Used with select query
2. executeUpdate( ):
   public int executeUpdate( ) throws SQLException
   Used with insert, update, delete, alter table etc.
3. execute( ):
   public boolean execute( ) throws SQLException
   Generally used with multiple results are generated.
   Also used with Create table query.

Example :

```
CallableStatement cst;
cst = con.prepareCall("{call remove( ? )}" );
cst.setString(1, "Dhruvi");
cst.executeUpdate( );
```

Define the call to the Database procedure

Procedure with no parameters.
{ call procedure_name }

Procedure with input parameters.
{ call procedure_name(?, ?, ...) }

Procedure with an output parameter.
{ ?= call procedure_name }

Procedure with input and output parameters.
{ ? = call procedure_name(?, ?, ...) }

Define the call to the Database procedure

Procedure with no parameters.
{ call procedure_name }

Procedure with input parameters.
{ call procedure_name(?, ?, ...) }

Procedure with an output parameter.
{ ?= call procedure_name }

Procedure with input and output parameters.
{ ? = call procedure_name(?, ?, ...) }

Using Database Transactions

- When a database is updated, by default the changes are permanently written (or committed) to the database.
- However, this default behavior can be programmatically turned off.
- If autocommitting is turned off and a problem occurs with the updates, then each change to the database can be backed out (or rolled back to the original values).
- If the updates execute successfully, then the changes can later be permanently committed to the database. This approach is known as transaction management.

Using Database Transactions

- The default for a database connection is autocommit; that is, each executed statement is automatically committed to the database.
- Thus, for transaction management you first need to turn off autocommit for the connection by calling setAutoCommit(false).
- Typically, you use a try/catch/finally block to properly handle the transaction management.
- First, you should record the autocommit status.
- Then, in the try block, you should call setAutoCommit(false) and execute a set of queries or updates.
- If a failure occurs, you call rollback in the catch block; if the transactions are successful, you call commit at the end of the try block.
- Either way, you reset the autocommit status in the finally block.

Using Database Transactions

The default for a database connection is autocommit; that is, each executed statement is automatically committed to the database.

Thus, for transaction management you first need to turn off autocommit for the connection by calling setAutoCommit(false).

Typically, you use a try/catch/finally block to properly handle the transaction management.

First, you should record the autocommit status.

Then, in the try block, you should call setAutoCommit(false) and execute a set of queries or updates.

If a failure occurs, you call rollback in the catch block; if the transactions are successful, you call commit at the end of the try block.

Either way, you reset the autocommit status in the finally block.

THANK YOU