Lab 9: JDBC

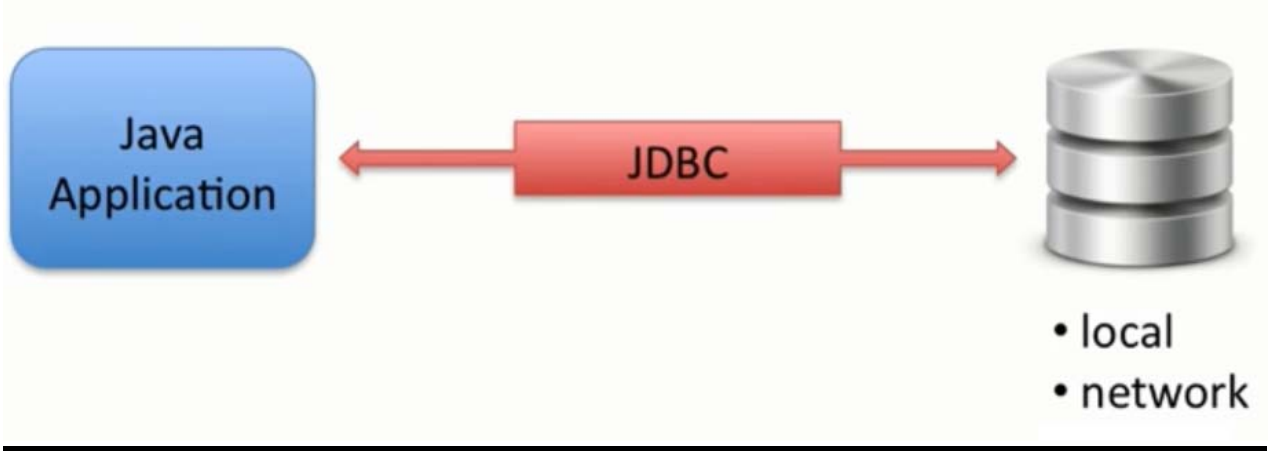
Objectives:

At the end of this lab, you should be able to

- Use JDBC to read, write data to Oracle database
- Build GUI that interacts with database.

Introduction:

Java Database Connectivity (JDBC) is an application programming interface (API) which allows the programmer to connect and interact with databases, the database might be saved locally or on network. It provides methods to query and update data in the database through update statements like SQL's CREATE, UPDATE, DELETE and INSERT and query statements such as SELECT. Additionally, JDBC can run stored procedures.



- It supports a large number of databases like Oracle, MySql, SQLServer, Sysbase, DB2,..etc.
- To connect to a specific database type, you need to use JDBC driver that is responsible of converting API calls to low level calls for that specific database, the driver is usually is normally provided by database vendor.

JDBC API

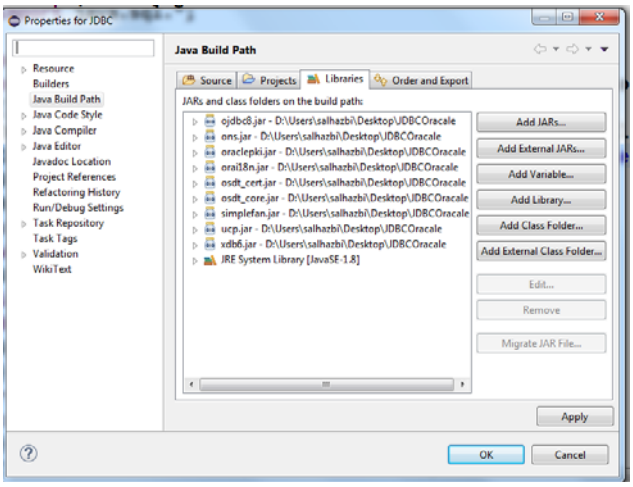
The main Java classes that implement JDBC:
-java.sql.DrvierManager.
java.sql.Connection
java.sql.Statement
java.sql.ResultSet
javax.sql.DataSource**.**

They are defined in the packages: java.sql and javax.sql.

Installing and using Oracle JDBC driver in Eclipse

We need to use Oracle JDBC driver, it can be downloaded from
http://www.oracle.com/technetwork/database/application-development/jdbc/downloads/index.html
(For our lab, it is already on Blackboard)

To add them to the project (your project in Eclipse), go to "project" menu, select "properties", then select "Java Build Path", select the tap "Libraries", then press "Add External JARs", select the files you download.



Writing JDBC program:

1- Create a connection to database
2- Create a statement object
3- Execute SQL query.
4- Process Result set.

Step 1: Create a connection to database:

In order to connect to a database, you need URL string to indicate location of the database, this might be locally on the same device, or maybe on network.
The syntax is

```
jdbc:<driver protocol>:<driver connection details>
```

for example with Oracle database

```
jdbc:oracle:thin@myserver:1521/demodb
```

Server name — Oracle Listener port — Service name

To create a connection, we also need user name and password :

```
Connection conn =
DriverManager.getConnection("jdbc:oracle:thin:@coestudb.qu.edu.qa:1521/STUD.qu.edu.qa","user", "pw");
```

Step 2: Create a statement object:

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

Step 3: Execute SQL query.

```
ResultSet rs = stmt.executeQuery("select ename, empno, sal from emp");
```

Step 4: Process Result set.

```
while (rs.next()) {
            String name = rs.getString(1);
            int number = rs.getInt(2);
            double salary = rs.getDouble(3);
            System.out.println(name + " " + number + " " + salary);
      }
```

- Method *ResultSet.next()* moves forward for one row, it returns true if there are more rows to process.
- We will use this method in a loop to process all retrieved data.
- To get specific field data from the retrieved data, we use *getXXX* methods, we can pass the name of the field, or a number corresponding the orders of the fields in SQL statement.
- We need to close the resultset and connection

```
rs.close();
conn.close();
```

```
import java.sql.*;
public class Test {
      public static void main(String args[]) throws SQLException {

      Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@
                        coestudb.qu.edu.qa:1521/STUD.qu.edu.qa","user", "pw");
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("select ename, empno, sal from emp");
            while (rs.next()) {
                  String name = rs.getString(1);
                  int number = rs.getInt(2);
                  double salary = rs.getDouble(3);
                  System.out.println(name + " " + number + " " + salary);
            }
            rs.close();
            conn.close();
      }
}
```

DML through JDBC

To insert, update, or delete records in database, instead of *executeQueury*, we use *executeUpdate*

Insert:

```
String sql=" insert into emp (empno,ename,sal,deptno)"
           +"values (333,'Khaled',4000,10)";
           stmt.executeUpdate(sql);
```

Update:

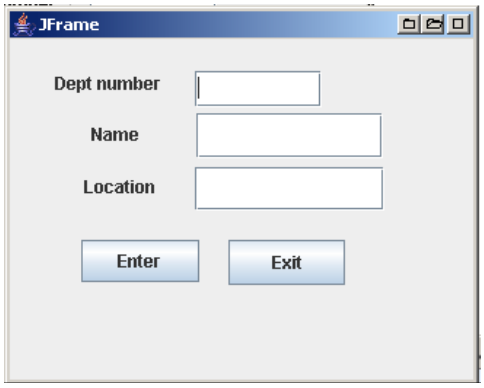
```
String sql=" update emp set sal=9000"+
                     " where empno=333";
           int affectedRows =stmt.executeUpdate(sql);
           System.out.println("Number of updated records "+affectedRows);
```

Delete:

```
String sql=" delete from emp where empno=333";
           int affectedRows=stmt.executeUpdate(sql);
           System.out.println("Number of updated records "+affectedRows);
```

Exercise 1

Using java swing components, developed friendly interface that reads department number, name, and its location from the user then insert this data to the table dept.

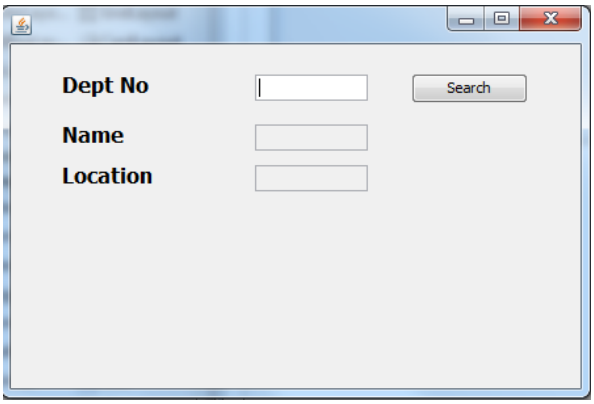


Hints:

- To reduce the complexity in the system, create a class "**Dbase"** to contain methods that interact with database, in above example, create method insertToDept that receives three parameters: int, String, String, and inserts them as a new record to Dept table.
- Use WindowBuilder to build the GUI.
- To exit, use method dispose( ).

Exercise 2

Using Java swing, develop GUI to search for a department using department number, your GUI should look as following
If no dept found, you should display dialog with the message "No dept. found".



Hints:

- To class Dbase in Ex1, add method getDeptData that receives an integer represents dept. number and returns an array of string contains name of the dept. and its location.