COMP2411

Lab 4 – Oracle

JAVA w/JDBC

What is JDBC?

- JDBC is a Java API (application program interface) for executing dynamic SQL statement.
- It consists of a set of classes and interfaces written in Java.
- It allows the programmer to send SQL statements to a database server for execution and retrieve query results (for SQL queries).
- It's provided for portability across database servers and hardware architectures.

Four Types of JDBC Drivers

- JDBC bridge driver.

  - This type uses bridge technology to connect a Java client to a third-party API such as Open Database Connectivity (ODBC).

- Native API (partly Java driver).

  - This type of driver wraps a native API with Java classes.

- Network protocol (pure Java driver).

  - This type of driver communicates using a network protocol to a middle-tier server.

- Native protocol (pure Java driver).

  - This type of driver, written in Java, communicates directly with the database.

Oracle's JDBC Drivers

- JDBC OCI driver

  - This is a Type 2 driver that uses Oracle's native OCI interface. It is commonly referred to as the OCI driver.

- JDBC Thin driver

  -This is a Type 4, 100% pure Java driver for client-side use.
  (The driver we use in our lab)

- JDBC internal driver

  - This is a Type 2, native code driver for server-side use with Java code that runs inside the Oracle database's JServer JVM.

- JDBC server-side Thin driver

  - This is a Type 4, 100% pure Java driver for server-side used with Java code that runs inside the Oracle database's JServer JVM.

A Simple JDBC Application

```
import java.io.*;
import java.io.Console;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;

public class simpleApplication
{
    public static void main(String args[]) throws SQLException, IOException
    {
        Console console = System.console();
        System.out.print("Enter your username: ");    // Your Oracle ID with double quote
        String username = console.readLine();         // e.g. "98765432d"
        System.out.print("Enter your password: ");    // Password of your Oracle Account
        char[] password = console.readPassword();
        String pwd = String.valueOf(password);

        // Connection
        DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        OracleConnection conn =
            (OracleConnection)DriverManager.getConnection(
               "jdbc:oracle:thin:@studora.comp.polyu.edu.hk:1521:dbms",username,pwd);

        Statement stmt = conn.createStatement();
        ResultSet rset = stmt.executeQuery("SELECT EMPNO, ENAME, JOB FROM EMP");
        while (rset.next())
        {
            System.out.println(rset.getInt(1)
            + " " + rset.getString(2)
            + " " + rset.getString(3));
        }
        System.out.println();
        conn.close();
    }
}
```

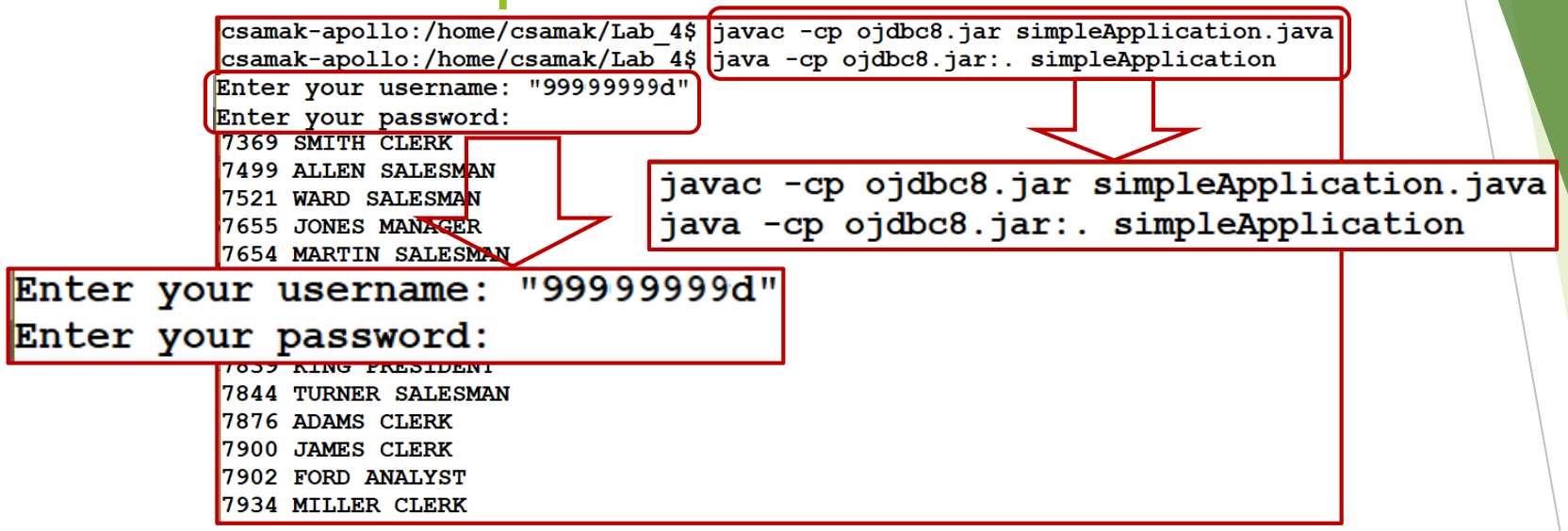
simpleApplication – "username"

```
Console console = System.console();
System.out.print("Enter your username: ");     // Your Oracle ID with double quote
String username = console.readLine();          // e.g. "98765432d"
System.out.print("Enter your password: ");     // Password of your Oracle Account
char[] password = console.readPassword();
String pwd = String.valueOf(password);
```

- For example, Oracle A/C ID is *98765432d*, when log in to the Oracle, it is required to use double quote with the ID.
- In the Java program, when you enter your username, please include the double quote.

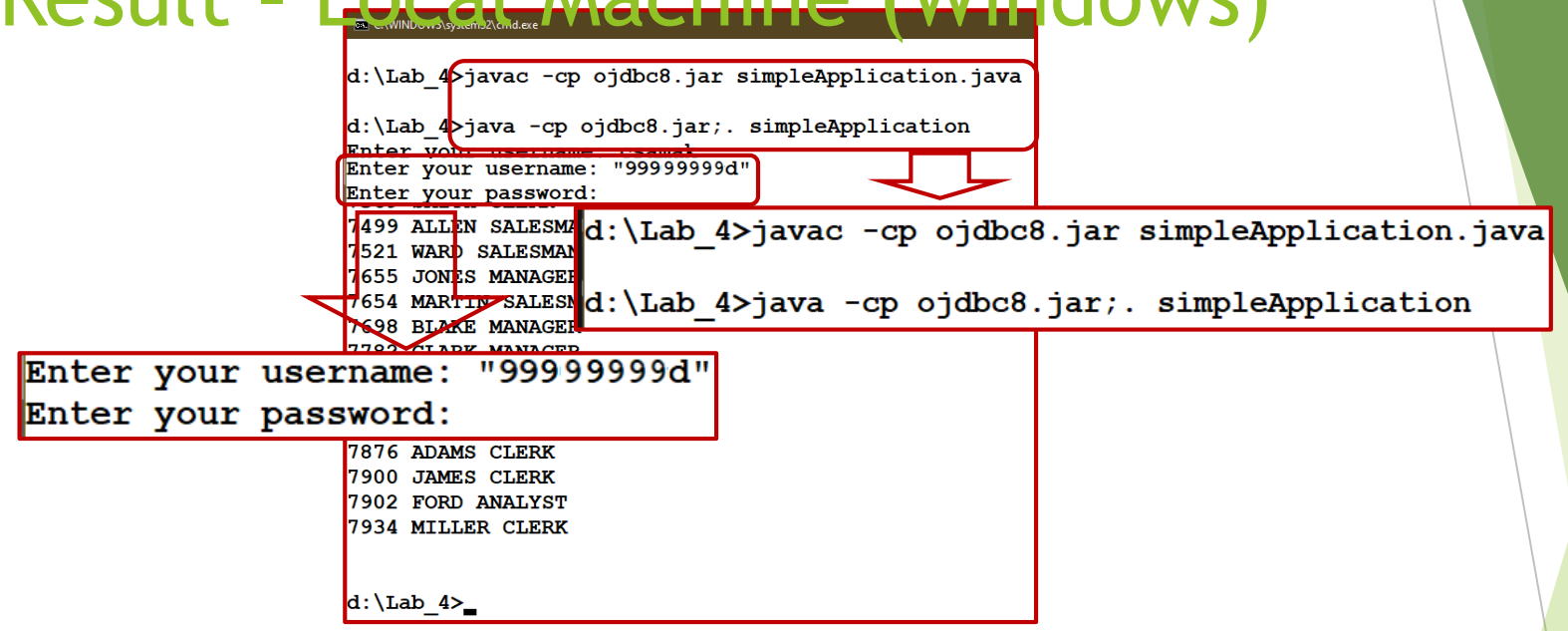
```
Enter your username: "98765432d"
Enter your password: <Your Oracle Password>
```

Execution Result – Department's Server



- Use Putty to connect to Department's server.
- Create a directory under your home directory and name it *Lab_4*.
- Place all the lab materials under that directory.
- Compile the java file (*simpleApplication.java*) with the use of "**classpath**" that points to the JDBC driver file (*ojdbc8.jar*).
- When execute the application, it is also necessary to use the "**classpath**" that points to the JDBC driver file and the current directory (use " **: .** ").

Execution Result – Local Machine (Windows)



- Create a folder and name it *Lab_4* in your computer (*for example, create the folder under Drive-D*).
- Place all the lab materials under that folder.
- Compile the java file (*simpleApplication.java*) with the use of "**classpath**" that points to the JDBC driver file (*ojdbc8.jar*).
- When execute the application, it is also necessary to use the "**classpath**" that points to the JDBC driver file and the current directory (use " **; .** ").

Making The Connection

- Import packages:

```
import java.io.*;
import java.io.Console;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.sql.*;
```

- Register JDBC driver:

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

- Formulate database URL:

```
DriverManager.getConnectioin(String url, String username, String password)
url = jdbc:oracle:driver:@database:port:sid
```

OR

```
DriverManager.getConnection(String url)
url = jdbc:oracle:driver:username/password@database:port:sid
```

- Disconnect from the database:

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

Creating JDBC Statements

- A Statement object is a class which sends your SQL statement to the DBMS.
- It takes an instance of an active connection to create a Statement object.

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

- Pass SQL statement on to the DBMS. For SELECT statement, the method to use is executeQuery. For statements that create or modify tables, the method to use is executeUpdate.

```
stmt.executeQuery("SELECT * FROM EMP");
```

OR

```
stmt.executeUpdate("CREATE TABLE COFFEES " +
   " (COF_NAME VARCHAR(32), SUP_ID INTEGER, PRICE FLOAT, " +
   " SALES INTEGER, TOTAL INTEGER)");
```

Retrieve Data From Database

- JDBC returns results in a ResultSet object, so we need to declare a reference of the class ResultSet to hold our results.

```
ResultSet rset = stmt.executeQuery("SELECT EMPNO,ENAME,JOB FROM EMP");
```

- Using the method next();

  1. The variable rset contains the rows of results retrieved from EMP.
  2. The method next() moves what is called a cursor to the next row and makes that row (called the current row) the one upon which we can operate.
  3. The cursor is initially positioned above the first row of a ResultSet object.

- Using the getXXX methods;

  We use the getXXX method of the appropriate type to retrieve the value in each column. For example:

```
rset.getInt(1);           // Use column number;
rset.getString("ENAME"); // Use column name;
```

  * Note: the column number refers to the column number in the result set, not in the original table. Column number starts from 1.

Using PreparedStatements

- When you want to execute a Statement object many times, the execution time will normally be shortened if a PreparedStatement object is used instead. It is derived from the more general class, Statement.
- The PreparedStatement object contains not just an SQL statement, but an SQL statement that has been precompiled.
- When PreparedStatement is executed, the DBMS can just run the PreparedStatement's SQL statement without having to compile it first.
- Create a PreparedStatement object:

```
PreparedStatement updateSales =
 conn.prepareStatement("UPDATE EMP SET SAL = ? WHERE ENAME = ?" );
```

- Supply values for PreparedStatement parameters:

  SetXXX methods:

```
updateSales.setInt(1, 1500);
updateSales.setString(2, "MARTIN");
```

- It can be executed with the following: `stmt.executeUpdate();`

PreparedStatement cont'd

- Using a loop to set values.

```
PreparedStatement updateSal;

  String updateString = "UPDATE EMP " + "SET SAL = ? WHERE ENAME = ?";
  updateSal = conn.prepareStatement(updateString);

  int[] salForUpdate = {1500,1750,2000,2250,2500};
  String enameForUpdate = {"SMITH","WARD","JONES","KING","FORD"};
  int len = enameForUpdate.length;

  for(int i=0;i<len;i++)
  {
     updateSal.setInt(1,salForUpdate[i]);
     updateSal.setString(2,enameForUpdate[i]);
     updateSal.executeUpdate();
  }
```

JDBC Transactions

- JDBC allows SQL statements to be grouped into a single transaction.
- Transaction control is performed by the Connection object, default mode is auto-commit, i.e., each SQL statement is treated as a transaction.
- We can turn off the auto-commit mode with conn.setAutoCommit(false) and turn it back on with conn.setAutoCommit(true);
- Once auto-commit is off, no SQL statement will be committed until an explicit commit is invoked: conn.commit().
- At this point, all changes done by the SQL statements will be made permanent in the database.

JDBC Transaction cont'd

- If we don't want certain changes to be made permanently, we can issue conn.rollback(); Any changes made since the last commit will be ignored – usually rollback is used in combination with Java's exception handling ability to recover from unpredictable errors.
- An Example:

```
conn.setAutoCommit(false);
Statement stmt = conn.createStatement();
stmt.executeUpdate("INSERT INTO DEPT VALUES(50,'HRO','NEW YORK')");
conn.rollback;
stmt.executeUpdate("INSERT INTO DEPT VALUES(50,'HRO','WASHINGTON')");
conn.commit();
conn.setAutoCommit(true);
```

Handling Errors with Exceptions

- Programs should recover and leave the database in a consistent state.
- In Java, statements which are expected to "throw" an exception or a warning are enclosed in a try block.
- If a statement in the try block throws an exception or warning, it can be caught in one of the corresponding catch statements.
- Example:

```
PreparedStatement stmt2 = conn.prepareStatement("INSERT INTO SALGRADE VALUES(6, 10000, 12500)");

try{
   stmt2.executeUpdate();
}  catch (SQLException e) {
   System.out.println("Record was not added! Error!");
   while(e!=null) {
       System.out.println("message: "+e.getMessage());
       e = e.getNextException();
   }
}
```

Stored Procedures

- A stored procedure is a group of SQL statements that form a logical unit and perform a particular task.
- Stored procedures are used to encapsulate a set of operations or queries to execute on a database server.
- Stored procedures can be compiled and executed with different combination of input, output parameters.
- Create a simple stored procedure in SQL:

```
CREATE PROCEDURE SHOW_STAFF
AS
SELECT ENAME,JOB,MGR,HIREDATE
FROM EMP ORDER BY ENAME;
```

Stored Procedures cont'd

- Calling a stored procedure from JDBC:

```
String createProcedure = "CREATE PROCEDURE SHOW_STAFF " +
                         "AS SELECT ENAME,JOB,MGR,HIREDATE " +
                         "FROM EMP ORDER BY ENAME";

Statement stmt = conn.createStatement();
stmt.executeUpdate(createProcedure);

CallableStatement cs = conn.prepareCall("{call SHOW_STAFF}");
ResultSet rset = cs.executeQuery();
```

- A CallableStatement object contains a call to a stored procedure, it does not contain the stored procedure itself.
- When the driver encounters "{call SHOW_STAFF} ", it will translate this escape syntax into the native SQL used by the database to call the stored procedure named SHOW_STAFF.