Connecting to Oracle with Java

Developing JDBC Applications

- The following basic steps are involved in developing JDBC applications:

1. **Import the JDBC classes (java.sql.*)**
2. Load the JDBC drivers.
3. Connect to the database.
4. Interact with the database using JDBC.
5. Disconnect from the database.

Importing the JDBC classes

- import java.sql.*;
- import java.io.*;
- import java.lang.*;
- import oracle.jdbc.driver.*;
- import java.sql.CallableStatement;
- import java.sql.Types;

Steps

1. Import the JDBC classes (java.sql.*)
2. **Load the JDBC drivers.**
3. Connect to the database.
4. Interact with the database using JDBC.
5. Disconnect from the database.

Loading the drivers

- A Java program can load several JDBC drivers at any time. This allows for the possibility of the program interacting with more than one database running on possibly different servers.

- The syntax to load Oracle JDBC drivers is:

```
Class.forName("oracle.jdbc.driver.OracleDriver")
```

Steps

1. Import the JDBC classes (java.sql.*)
2. Load the JDBC drivers.
3. **Connect to the database.**
4. Interact with the database using JDBC.
5. Disconnect from the database.

Connecting to the Database

- DriverManager class
- getConnection method forms the connection

```
Connection conn
conn = DriverManager.getConnection(
               url, username, password);
```

where url is of the form:

```
jdbc:oracle:drivertype:user/password@database
```

Thin client Connection URL

- The @database is
- either
  - host:port:sid // Use this with Oracle Express or 11g
- or
  - host:port/servicename // Use this with 12c

The connection

```
String servername, portnumber, sid, url;
servername = "localhost";
portnumber = "1521";
sid = "xe";
url = "jdbc:oracle:thin:@" + servername + ":" +
portnumber + ":" + sid;
```

Developing JDBC Applications

- The following basic steps are involved in developing JDBC applications:

1. Import the JDBC classes (java.sql.*)
2. Load the JDBC drivers.
3. Connect to the database.
4. **Interact with the database using JDBC.**
5. Disconnect from the database.

Ways of interacting

- To run a function from a Java program, use
  - `CallableStatement`
- Prepare the CallableStatement using `prepareCall`
- Example – Adding a student

```
CallableStatement stmt = null;
stmt = conn.prepareCall ("{? = call ADDSTUDENT(?,?,?,?)}");
```

- This sets up the statement as needing 5 parameters. Then later

```
stmt.execute();
```

Setting up the parameters

- Parameters are set up sequentially in the order they occur.
- Output parameters are registered with an SQL type:
  - stmt.registerOutParameter(1,Types.VARCHAR);
- Input parameters are set with a Java type:
  - stmt.setString(2, prog);
  - stmt.setInt(3,stage);
  - stmt.setString(4, sname);
  - stmt.setString(5, saddr);

Getting the results

- The stmt object has its five parameters, the first of these is the VARCHAR containing the student number.
- To retrieve this after execution:

```
sno = stmt.getString(1);
```

Steps

1. Import the JDBC classes (java.sql.*)
2. Load the JDBC drivers.
3. **Connect to the database.**
4. Interact with the database using JDBC.
5. Disconnect from the database.

Disconnect

- Before leaving, close the statement and the connection:

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

A sample java program

- The example AddAStudent.java is in webcourses.
- This program uses very basic command line data and a compiler called javac with a small footprint.
- I have also included AddAStudent.bat that is a script that will run from the command line in windows:
  - You must run it from the directory the program is in
  - You must have your path set to find javac.exe
  - You must have included the ojdbc7.jar in your directory.
- You can do this from any java program but be aware of firewalls.