INTRODUCTION TO
JDBC

- Revised spring 2004 -

1 What is JDBC?

Java Database Connectivity (JDBC) is a package in the Java programming language and consists of several Java classes that deal with database issues such as connections, queries etc. Included in the Java toolkit is the *JDBC-ODBC Bridge* driver but there are other specific drivers available from different database manager manufacturers. In this introduction we'll be using the DB2 JDBC driver from IBM named: *COM.ibm.db2.jdbc.app.DB2Driver*. This is important since different drivers do not always act the same way.

To be able to compile and execute a Java program that uses the JDBC framework you need to have two things. First, you need to have the *Java Standard Development Toolkit* (SDK) installed. This environment is free of charge and could be downloaded from: http://java.sun.com/j2se/1.4.2/download.html Secondly, you need to have a ODBC or a JDBC driver installed that corresponds to the database management system (DBMS) you are using or plan to use. These drivers are often supplied by the DBMS manufacturers.

Just similar to any Java application a JDBC enabled Java application is compiled through the Java compiler *javac*. The compiler creates machine readable (executable) code out of your Java source code which could be executed through the virtual machine by the *java* command.

2 An example (JDBCAppa):

On the on the local network path: \\db-srv-1\studkursinfo\IS4 vt2004\myjdbctest you'll find the file *JDBCAppa.java*. This is the complete source code of this example and there you'll see all syntax used in this introduction in its context.

1. First of all, create a new folder (preferably on your m: drive) and name it something convenient like for example *myJDBC*.
2. Copy the source file *JDBCAppa.java* to your new folder.

Now, the *JDBCAppa.java* file is not compiled yet, as you'll see if you open it in a text editor like for example notepad. It's is still in a high level programming language, which is easily readable and meaningful for the human eye. This means that the application cannot be executed since what tends to be readable for a computer is garble for humans and vice versa.

3. Compile the *.java* file by using the *javac* compiler to create a new *.class* file. Do this by placing yourself in the newly created folder in a *Windows Command Prompt* (in this case *m:\myJDBC*) and issue the command: **`javac JDBCAppa.java`**

As you'll see, this procedure created the file *JDBCAppa.class*. This is the low level language and virtual machine specific instructions understood by the computer (and not by humans). Hopefully you now have two versions of the application. One that you are able to read, the other meaningful for the computer. Execute the application by using the virtual machine caller command: **`java JDBCAppa`**. Every time you change something in the source code you have to recompile the *.class* file to see the effects when running the application.

2.1 Explanation of the source code

Open up *JDBCAppa.java* to view the application source code. Exactly as the compiler and the virtual machine (VM), we'll go through the file, reading from the top.

```
import java.sql.*
…
```

This is a statement to include compiled Java methods and constructs from other packages. The star "*****" states that we shall include all classes in the *java.sql* package path. This is the package that includes all JDBC components that we are interested in.

```
…
static protected Connection con;
…
private String URL = ”jdbc:db2:sample”;
private String userID = “”;
private String driver = “COM.ibm.db2.jdbc.app.DB2Driver”;
private String password = “”;
…
```

This section creates a lot of variables. *con* is the variable that is going to manage out database connection, *URL* is the path of our database, *driver* specifies where the driver details are located and *userID* and *password* are giving the required information to log on to the database. After the variables are created, the compiler/VM call the *main* method which in turn call out to the *connect*, *select* and *update* methods.

2.1.1 The connect method

```
…
class.forName(getDriver());
…
```

This method loads the specified driver.

```
…
con = DriverManager.getConnection(getURL(), getUserID(), password);
…
```

This creates a connection instance and attaches this to the *con* handle created earlier. The connection instance connects to the DB2 database by the help of the *URL*, *userID* and *password* variables.

```
…
con.setAutoCommit(false);
…
```

This specifies that all changes made in the database are not inserted until an explicit *commit* command is issued. If *autocommit* is turned on, every query is committed directly after it has been executed.

2.1.2 The select method

At first the variables query (String), rs (ResultSet) and stmt (Statement) are declared. Query holds the SQL string that's specifying what we are looking for, rs is the variable that is going to keep the query results and stmt is the variable that executes the query against the database connection.

```
…
query = “SELECT empno, firstnme FROM employee;”;
…
```

This specifies our query which will collect all employment numbers and first names of all employees registered in the database.

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

The *stmt* handle is associated with a statement object that in turn is linked with the database connection.

```
…
rs = stmt.executeQuery(query);
…
```

This statement executes the SQL query specified in the string query against the connection specified in *stmt*. The result of the query is put in *rs*.

```
…
while(rs.next()){
      System.out.print(“ empno= ”+rs.getString(“empno”));
      System.out.print(” “);
      System.out.println(“ firstname= ”+rs.getString(“firstnme”));
}
…
```

This WHILE-loop iterates through the collected rows in the *data collection rs* until the last row is fetched. The method *next* returns false when the collection is empty, ending the WHILE-loop. For each iteration the metod *getString* retrieves the value of the column specified in its argument. For example *rs.getString(“firstnme”)* retrieves the value of the column named *firstnme* in the fetched row. It is also possible to use other methods to collect values of different data types like Integers or Booleans. For example specifying *rs.getFloat(6)* would retrieve the sixth column in the row and treat the collected value as a Float. This can be seen in the last part of the select method in *JDBCAppa* where a non-decimal number is fetched by the *getInt* method.

2.1.3 The update method

This is a little different then the previous method since we’re not retrieving anything, only changing old or inserting new data. In this case we’re also interested in working with dynamic parameters that are changed during the execution of the program. To do this we use the *PreparedStatement* class instead of *Statement*. The main difference is that *PreparedStatement* uses ‘*?*’ signs as dynamic parameters. Consider the following:

```
…
query = “UPDATE employee SET firstnme = ‘SHILI’ WHERE empno =?;”;
…
```

This creates an ordinary SQL string with the exchangeable parameter ‘*?*’.

```
…
stmt = con.preparedStatement(query);
…
```

This does exactly the same as when declaring the statement in the select method, although this is a *PreparedStatement.*

```
…
stmt.setString(1,param1);
…
```

This is interesting since it changes the first ‘*?*’ parameter of the query specified in *stmt* to the content of the string *param1*. The first integer argument specifies which parameter

number should be changed. For example *setString(5,"Mike")* would have changed the fifth question mark with the text *Mike*.

```
…
stmt.executeUpdate();
…
```

This would execute the update in the database. Note that since we don't collect any data we don't need any *ResultSet* to capture it in.

3 Additional information:

JDBC Tutorial: http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html
Java Basics: http://java.sun.com/docs/books/tutorial/getStarted/cupojava/win32.html
java.sql Documentation: http://java.sun.com/j2se/1.3/docs/api/java/sql/package-summary.html

4 Writing your first JDBC application

It's greatly recommended to use your own folder under the *m:* drive as a base when creating the application. Try to reuse as much of the *JDBCAppa.java* as possible, modify it and make adjustments rather then starting from scratch. A few points to keep in mind when doing this:

1. The name of the file and the class must be the same. Renaming the class = renaming the file and vice versa.
2. The database path and login information must be correct in the application.