INTRODUKTION TILL JDBC

Vad är JDBC?

JDBC står för *Java DataBase Connectivity*. JDBC ingår i Java och består av en del klasser som har hand om databasfunktionalitet. Med Java följer JDBC-ODBC Bridge driver, men man kan installera även andra DBMS specifika drivers om man så vill. I den här introduktionen kommer vi att använda IBM:s DB2 JDBC driver (**COM.ibm.db2.jdbc.app.DB2Driver**). Alternativt kan man använda den JDBC-ODBC Bridge driver.

Vad kräver det?

För att kunna kompilera och köra ett JDBC Java program (kallat *JDBC program* i fortsättningen) måste man:

- ha JDK installerat på datorn.
- ha ODBC, eller en DBMS specifik driver installerad på datorn

Ett JDBC program kompileras med java kompilatorn (kommandot **javac**). När programmet är färdigtkompilerat kan man köra det genom att köra den skapade **.class** filen. Det gör man med kommandot **java** och *klassnamnet.* JDBC kan användas i alla sorter Java program. Man kan skapa en applet eller en servlet som använder JDBC. JDBC delen är oberoende av vilken typ av Java program man skriver.

JDBCAppa exemplet

JDBCAppa är namnet på JDBC exemplet som finns att ladda ner från

\\Db-srv-1\StudKursInfo\IS4 vt2003\MyJDBCTest
(man hittar dit via "Network Neighborhood")

Där finner man följande fil:

JDBCAppa.java

Skapa en egen katalog och kopiera dit programmet!

JDBCAppa.java är det okompilerade JDBC programmet. Programmet kan kompileras med java kompilatorn (**javac**) och köras med kommandot **java JDBCAppa**. Programmet kan också ta emot två parametrar, *username* och *password*, som kan användas för databas kopplingen.

Här är en förklaring av de viktigaste JDBC kommandon som finns i **JDBCAppa.java**:

I början av programmet finns följande rad:

```
import java.sql.*;
```

Det är **java.sql** klassbiblioteket som innehåller alla JDBC komponenter.

I JDBCAppa klassen finner man först en del variabeldeklarationer:

```
// DB connection variable
static protected Connection con;

// DB access variables
private String URL = "jdbc:db2:sample";
private String userID = "";
private String driver = "COM.ibm.db2.jdbc.app.DB2Driver";
private String password = "";
```

Con är variabeln som kommer hålla i databaskopplingen, resten kommer att användas för att få en databas koppling. **URL** innehåller databasnamnet som är associerat till drivern som finns i variabel **driver**.

I slutet på **JDBCAppa.java** finns metoden **main()**. Det är denna metod som körs först, när man kör programmet. Den gör följande:

Först skapar den en instans av klassen **JDBCAppa**. Sedan kontrollerar den de inkommande parametrarna. Till slut anropar **main()** metoderna **connect(), select()** och **update()** i den nya instansen av **JDBCAppa** (kallad **t**).

Metoden **connect()** gör följande:

Först laddar den rätt driver med kommandot

```
Class.forName(getDriver());
```

Sedan skapar den en koppling och lägger den i variabel **con** med kommandot

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

Till slut sätter den databaskopplingens **AutoCommit** till **false**:

```
con.setAutoCommit(false);
```

Det betyder att alla ändringar som görs via den databaskopplingen kommer inte att registreras i databasen förrän man gör **commit**.

Metoden **select()** gör följande:

Först deklareras variablerna **query, rs** och **stmt**, som kommer att användas för att exekvera en SELECT sats och ta emot resultatet. SELECT satsen placeras i variabeln **query**:

```
query = "SELECT empno, firstnme from employee;";
```

Variabeln **stmt** kopplas till en **Statement** som är associerat till kopplingen **con**:

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

Variabeln **rs** tilldelas resultatet när SELECT satsen i **query** körs med hjälp av **stmt**:

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

Metoden **executeQuery()** används när man har en SQL sats som returnerar något resultat, som till exempel SELECT satser.

Sedan gäller det bara att gå igenom resultatet och skriva ut det:

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

Metoden **next()** går framåt i resultatet så att nästa rad blir den aktuella. Om det inte finns flera rader returneras **false**. Metoden **getString()** returnerar värdet som finns i aktuell rad och kolumn. Kolumnen kan anges antingen med kolumnnamnet eller med kolumnposition. Alltså **rs.getString(1)** skulle kunna användas istället för **rs.getString("empno")).**

Istället för metoden **getString()** kan man använda en av följande metoder beroende på kolumnens datatyp:

getDate()	**getInt()**	**getByte()**
getDouble()	**getString()**	**getBoolean()**
getFloat()	**getLong()**	

I sista delen av metoden **select()** används metoden **getInt()** för att hämta ett heltal. Resten av logiken där är exakt samma som tidigare.

Metoden **update()** jobbar istället med någonting som kallas för **PreparedStatement**. Skillnaden från en vanlig **Statement** (som användes i metoden **select()**) är att man kan använda parametrar i SQL satserna. En parameter i en SQL sats markeras med ett frågetecken. Frågetecknet kommer så småningom att bytas ut mot det riktiga värdet. Variabeln **query** tilldelas en SQL sats med en parameter:

```
query = "UPDATE employee set firstnme = 'SHILI' where empno = ? ;";
```

Variabeln **query** kan sedan användas för att skapa en instans av klassen **PreparedStatement**, som placeras i variabeln **stmt**:

```
stmt = con.prepareStatement(query);
```

Nu kan parametern bytas ut mot värdet i variabeln **param1**:

```
stmt.setString(1, param1);
```

setString() metoden byter ut en parameter med en variabel av typ **String**. Den första parametern i metoden **setString()** identifierar positionen av parametern i SQL satsen som ska bytas ut. Alltså, kommandot ovan byter ut den första parametern (första frågetecknet) mot variabeln **param1**.

Då är det dags att exekvera SQL satsen. Det gör man med metoden **executeUpdate()**:

```
stmt.executeUpdate();
```

Metoden **executeUpdate()**, till skillnad från metoden **executeQuery()**, returnerar inte något resultat. Den kan användas för DDL satser och DML satser förutom SELECT satser. Båda metoder finns för både **Statement** och **PreparedStatement**. Skillnaden är att för **Statement** måste man ha en **String** variabel med SQL satsen som parameter. För **PreparedStatement** har man definierat SQL satsen innan man exekverar den.

Man kan också använda **PreparedStatement** för SQL satser som inte har parametrar:

```
stmt = con.prepareStatement("ROLLBACK work;");
stmt.executeUpdate();
```

Programmet **JDBCAppa.java** kan man kompilera med kommandot **javac JDBCAppa.java** och efteråt köra det med kommandot **java JDBCAppa**.

Här är programmets resultat:

```
Received results:
 empno= 000010   firstname= CHRISTINE
 empno= 000020   firstname= MICHAEL
 empno= 000030   firstname= SALLY
 empno= 000050   firstname= JOHN
 empno= 000060   firstname= IRVING
 empno= 000070   firstname= EVA
 empno= 000090   firstname= EILEEN
 empno= 000100   firstname= THEODORE
 empno= 000110   firstname= VINCENZO
 empno= 000120   firstname= SEAN
 empno= 000130   firstname= DOLORES
 empno= 000140   firstname= HEATHER
 empno= 000150   firstname= BRUCE
 empno= 000160   firstname= ELIZABETH
 empno= 000170   firstname= MASATOSHI
 empno= 000180   firstname= MARILYN
 empno= 000190   firstname= JAMES
 empno= 000200   firstname= DAVID
 empno= 000210   firstname= WILLIAM
 empno= 000220   firstname= JENNIFER
```

```
 empno= 000230   firstname= JAMES
 empno= 000240   firstname= SALVATORE
 empno= 000250   firstname= DANIEL
 empno= 000260   firstname= SYBIL
 empno= 000270   firstname= MARIA
 empno= 000280   firstname= ETHEL
 empno= 000290   firstname= JOHN
 empno= 000300   firstname= PHILIP
 empno= 000310   firstname= MAUDE
 empno= 000320   firstname= RAMLAL
 empno= 000330   firstname= WING
 empno= 000340   firstname= JASON

Retrieve the number of rows in employee table...
There are 32 rows in employee table.

Update the database...

Retrieve the updated data from the database...
 empno= 000010   firstname= SHILI

Rollback the update...
Rollback done.
```

Att skriva ett eget JDBC program!

Det rekommenderade arbetssättet är att man skapar en katalog (till exempel **NyJDBCProg**) och kopierar dit **programmet JDBCAppa.java** som man sedan kan modifiera. Man kan även titta på gamla tentor och de lösningsförslag till dem om man vill se andra exempel eller hämta "inspiration" för att skriva ett eget program.

Ytterligare 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 dokumentation:
http://java.sun.com/products/jdk/1.1/docs/api/Package-java.sql.html