gds2 Gottlieb-Daimler-Schule 2
Technisches Schulzentrum Sindelfingen
mit Abteilung Akademie für Datenverarbeitung

1 Hintergründe

Java **D**ata **B**ase **C**onnectivity (JDBC) ist eine Programmierschnittstelle (API), über die man in Java Daten aus einer Datenbank verarbeiten kann. JDBC besteht aus einer Menge von Klassen und Schnittstellen, die im Paket java.sql zusammengefasst sind.

Mit JDBC ist die Entwicklung von Anwendungen für eine Vielzahl von SQL-Datenbanken möglich, ohne dass dabei die Besonderheiten der einzelnen DBMS (Datenbank Management Systeme) berücksichtigt werden müssen. So kann eine JDBC-Anwendung, die für ORACLE entwickelt wurde, durch einfachen Austausch des Treibers mit einer MySQL-Datenbank arbeiten.

JDBC hilft bei der Erstellung von Anwendungen mit DB-Anbindung, indem es folgende drei elementare Schritte in diesem Zusammenhang unterstützt:

1. Verbindung zu einer Datenquelle, wie z.B. zu einer Datenbank herstellen.
2. Abfrage- und Aktualisierungsbefehle an das DBMS senden.
3. Die Abfrageergebnisse vom DBMS abrufen und verarbeiten.

Die JDBC API enthält zu diesem Zweck entsprechende Klassen:

- Klassen zur Realisierung von Datenbanktreibern, die den Anschluss von Java-Anwendungen an DBMS wie Oracle, MySQL, MS-Access etc. ermöglichen.
- Die Klassen des Pakets `java.sql` bestehend aus dem
  Manager für diese Treiber,
  Interfaces als Schnittstellen zu den Treibern
  Hilfsklassen für Datum, Uhrzeit und JDBC-Typen

Treiber sind datenbank- und herstellerabhängige Java-Klassen, die sich aber als JDBC-Interfaces dem Anwender einheitlich präsentieren. Sie dienen dem Zweck, Java-Programmen möglichst herstellerneutrale Programmierschnittstellen anzubieten. Damit soll ein Tausch des verwendeten DBMS ohne viel Aufwand ermöglicht werden.

2 Ausführen von SQL Befehlen mit JDBC

Im allgemeinen sind zur Ausführung von SQL Befehlen in einer Java Anwendung folgende Schritte notwendig:

1. Datenbank-Verbindungsinformationen bereitstellen (z.B. als Interface)
2. Verbindung zur Datenbank herstellen.
3. SQL Befehlsstring bereitstellen.
4. Ein `Statement` Objekt erzeugen.
5. Mit Hilfe des `Statement` Objektes den SQL Befehl ausführen.
6. Das Ergebnis des SQL Befehls als `ResultSet` (Ergebnismenge) entgegennehmen.
7. Die Daten aus dem `ResultSet` Objekt verarbeiten.
8. Die Verbindung zur Datenbank wieder schließen.

Auftretende Exceptions sollten auf jeden Fall immer bearbeitet werden.

2.1 Datenbank-Verbindungsinformationen bereitstellen

Man kann die Verbindungsinformationen direkt beim Verbindungsaufbau im DB-Verbindungsstring (s.u.) angeben. Um die Übersichtlichkeit und die Änderbarkeit zu erhöhen, ist es aber sinnvoll den Verbindungsstring aus symbolischen Konstanten zusammenzusetzen, die zentral in einem Interface deklariert sind.

Beispiel:

```
public interface Daten
{
  // Datenbank-Verbindungsdaten
  // Treiberklasse
  public static final String dbTreiber = "org.mariadb.jdbc.Driver";
  // Datenbankserver (DNS-Name oder IP Adresse angeben)
  public static final String host      = "localhost";
  // DBMS Server Port (Standardport MySQL: 3306)
  public static final String port      = "3306";
  public static final String db        = "u_itc_formular";   // Datenbankname
  public static final String user      = "kds";              // Datenbank-User
  public static final String passwd    = "kds2000";          // Passwort
  public static final String table     = "personen";         // Tabellenname
}
```

2.2 Verbindung zum DBMS herstellen (Connect)

Als erstes muss man eine Verbindung zu der Datenquelle herstellen, die man verwenden möchte. Prinzipiell bietet JDBC die Möglichkeit verschiedene Arten von Datenquellen zu benutzen. Neben einer direkten Verbindung zu einem DBMS können auch Verbindungen zu herkömmlichen Dateien und sogenannten „DataSource Objekten“ hergestellt werden.

Wir nutzen die Methode, bei der mit Hilfe der `DriverManager` Klasse eine direkte Verbindung zum DBMS Server hergestellt. Bei dieser Methode werden die Verbindungsdaten aus 2.1 benötigt.

Anmerkungen (siehe Bsp. unten):

1	Verweis auf `Connection` Objekt: repräsentiert die aufgebaute Verbindung
2	Die vorab installierte und im Build Path bekannt gegebene Treiberklasse muss eingebunden werden.
3	Der Verbindungsstring wird zusammengesetzt; hier: `"jdbc:mariadb://localhost:3306/u_itc_formular?user=kds&password=kds2000"`
4	Mit Hilfe der `DriverManager` Klasse und dem DB-Verbindungsstring wird versucht die Verbindung zum DBMS herzustellen.
5	Da der Verbindungsaufbau aus verschiedensten Gründen schief gehen kann, ist es für die Fehlersuche dringend notwendig, die Exceptions zu verarbeiten.
6	Das Connection Objekt wird für die Verwendung im weiteren Programmverlauf zurückgegeben. Der Aufrufer der Methode muss dafür sorgen, dass die Verbindung wieder geschlossen wird, sobald sie nicht mehr benötigt wird.

Beispiel:

```
   public class DBTabelleLesen implements Daten
   {
      public static Connection baueVerbindungAuf()
      {
1       Connection verbindung = null;
        try
        {
2         Class.forName(dbTreiber);

3         String s = "jdbc:mariadb://" + host + ":" + port + "/" + db + "?"
                     + "user=" + user + "&" + "password=" + passwd;

4         verbindung = DriverManager.getConnection(s);
        }
5       catch (ClassNotFoundException e)
        {
          System.out.println("Treiber nicht gefunden");
        }
        catch (SQLException e)
        {
          System.out.println("Connect nicht moeglich:" + e.getMessage());
        }

6       return verbindung;
      }
```

2.3 SQL Query ausführen und auswerten

Sobald erfolgreich ein `Connection` Objekt erzeugt wurde, kann mit Hilfe eines `Statement` Objektes ein SQL Befehl auf dem Server ausgeführt werden.

Anmerkungen (siehe Bsp. unten):

1	Das `Connection` Objekt wird hier als Parameter bereitgestellt
2	Ein SQL Query String wird im Bsp. fest im Programm angelegt.
3	Zur Ausführung des SQL Befehls wird ein `Statement` Objekt benötigt, das hier in einem try-mit-Ressource-Konstrukt erzeugt wird.
4	Mit Hilfe des `Statement` Objektes kann der SQL Befehl auf dem Server ausgeführt werden. Das Ergebnis des SELECT wird in Form eines `ResultSet` Objektes bereitgestellt. Das `ResultSet` enthält die Datensätze.
5	Das `ResultSet` besitzt einen Cursor, der zunächst vor dem ersten Datensatz steht. Die Methode `next()` setzt den Cursor auf den nächsten Datensatz im `ResultSet` und liefert true, solange der Cursor noch auf einen Datensatz verweist.
6	Die Spaltenwerte eines Datensatzes können mit typspezifischen get-Methoden ausgelesen werden. Anstelle der Spaltennamen können auch Nummern verwendet werden. (s.u.)

Beispiel:

```
1  public static void leseTabelle(Connection dbVerbindung)
   {
2    String sQuery = "select ArtikelNr, ArtikelName, Artikelgruppe,"
                         + " Einstandspreis, Lagerbestand from Artikel";
3    try (Statement stmt = dbVerbindung.createStatement())
     {
4      ResultSet rs = stmt.executeQuery(sQuery);

5      while (rs.next())
       {
6        String sArtikelNr = rs.getString("ArtikelNr");
         String sArtikelName = rs.getString("ArtikelName");
         String sArtikelgruppe = rs.getString("Artikelgruppe");
         double dEinstandspreis = rs.getFloat("Einstandspreis");
         int iLagerbestand = rs.getInt("Lagerbestand");
         if (sArtikelName.length() > 50)
           sArtikelName = sArtikelName.substring(0, 50);
         System.out.printf("%-6s %-50s %-13s %8.2f € %5d\n", sArtikelNr,
             sArtikelName, sArtikelgruppe, dEinstandspreis, iLagerbestand);
       }

     }
     catch (SQLException e)
     {
       System.out.println(e.getMessage());;
     }
   }
```

Alternative Auswertung der Datensätze:

```
       while (rs.next())
       {
         String sArtikelNr = rs.getString(1);
         String sArtikelName = rs.getString(2);
         String sArtikelgruppe = rs.getString(3);
         double dEinstandspreis = rs.getFloat(4);
         int iLagerbestand = rs.getInt(5);
         if (sArtikelName.length() > 50)
           sArtikelName = sArtikelName.substring(0, 50);
         System.out.printf("%-6s %-50s %-13s %8.2f € %5d\n", sArtikelNr,
             sArtikelName, sArtikelgruppe, dEinstandspreis, iLagerbestand);
       }
```

2.4 Beispiel einer einfachen DB Anwendung

Die oben beschriebenen Schritte werden im nachfolgenden Beispiel durch Aufruf der Methoden angewendet:

Einfache DB Anwendung:

```
import java.sql.*;

/**
 * @author stk
*/
public class Demo1_Start
{
      public static void main(String[] args) throws SQLException
      {
            Connection dbVerbindung = null;

            dbVerbindung = DBTabelleLesen.baueVerbindungAuf();

            if (dbVerbindung != null)
            {
                  DBTabelleLesen.leseTabelle(dbVerbindung);
                  dbVerbindung.close();
            }
      }
}
```

2.5 Metadaten eines ResultSet bestimmen

Zu einem ResultSet können auch Metainformationen über die zugrundeliegenden Daten abgerufen werden. Dazu gehören:

- Die Anzahl der Spalten im ResultSet
- Die Spaltennamen
- Die Java Datentypen der Spalten
- und vieles mehr (siehe Java Doku)

Einfache DB Anwendung:

```
      ResultSet rs = stmt.executeQuery(sQuery);
      ResultSetMetaData rsmd = rs.getMetaData();

      int iAnzahl = rsmd.getColumnCount();  // Anzahl der Spalten bestimmen
```