

3 Datenbankzugriff in Java

3.1 Erster einfacher Datenbankzugriff

Bemerkung:

In den folgenden Kapiteln werde ich nicht auf die bereits bestehenden Datenbanken *Kaffeeladen* zurückgreifen. Somit stehen diese Datenbanken unverändert zur Verfügung. Die Datenbankzugriffe in Java erfolgen auf Kopien *Kaffeehaus*.

Da alle Klassen zum Zugriff auf die JDBC-Schnittstelle im Paket *java.sql* liegen, muss dieses am Anfang des Programms importiert werden.

Die Datenbanken *Kaffeehaus1* verwendet folgende Daten:

<i>dbase</i>	(jdbc:mysql://localhost/) Kaffeehaus1
<i>benutzer</i>	ralph
<i>passwort</i>	ralph

Diese Daten werden in die entsprechenden Datenfelder *dbase*, *benutzer* und *passwort* gespeichert und im Konstruktor der Klasse *DBVerbindung* zusammengestellt:

```
import java.sql.*;

public class DBVerbindung
{
    private final String treiber = "com.mysql.jdbc.Driver";

    private String dBase = "jdbc:mysql://localhost/";
    private String benutzer;
    private String passwort;

    private Connection con;
    private Statement stmt;

    /** Konstruktor stellt die Daten zum Verbindungsaufbau zusammen.*/
    public DBVerbindung(String dBaseH, String benutzerH, String passwortH)
    {
        //Stellt die Daten zusammen, um eine Verbindung aufzubauen.
        dBase = dBase + dBaseH;
        benutzer = benutzerH;
        passwort = passwortH;
    }
}
```

Abbildung 3.1: Datenfelder und Konstruktor der Klasse *DBVerbindung*

Um auf eine Datenbank zugreifen zu können, muss eine Verbindung zu ihr hergestellt werden. Dazu wird zuerst der Datenbanktreiber geladen und ein Verbindungsobjekt erschaffen. Dieses Verbindungsobjekt bleibt während der ganzen Verbindung bestehen und dient als Lieferant für spezielle Objekte zur Abfrage und Veränderung der Datenbank.

Diese oben beschriebenen Aufgaben werden in der Methode *erstelleVerbindung()* durchgeführt.

```
public void erstelleVerbindung()
{
    try {
        //Laedt den Datenbanktreiber
        Class.forName(treiber);
        //Stellt die Verbindung her
        con = DriverManager.getConnection(dBase, benutzer, passwort);
        //Erzeugt ein Objekt fuer Abfragen und Aenderungen der Datenbank
        stmt = con.createStatement();
    }
    catch (ClassNotFoundException cnfe) {
        System.out.println(cnfe.toString());
    }
    catch (SQLException sqle) {
        System.out.println(sqle.toString());
    }
}
```

Abbildung 3.2: Die Methode *erstelleVerbindung()* in der Klasse *DBVerbindung*

Das Laden der Treiberklasse erledigt die Klasse *Class* durch den Aufruf der Methode *forName()*. Da wir hier den *mysql*-Datenbanktreiber verwenden, wird der Methode *forName()* der Parameter „*com.mysql.jdbc.Driver*“ übergeben, der in der Konstanten *treiber* gespeichert wird.

Nun kann dieser Treiber verwendet werden, um eine Verbindung zu einer Datenbank aufzubauen. Hierfür liefert die Methode *getConnection()* der Klasse *DriverManager* ein Verbindungsobjekt *con*, das die aktuelle Datenbanksitzung repräsentiert und bestimmte Anweisungsobjekte liefert. Dieser Methode werden drei Parameter übergeben, die allerdings abhängig von dem Aufbau und der Einstellung der Benutzerrechte der jeweilig verwendeten Datenbank sind.

Zum Schluss wird noch das *Statement*-Objekt *stmt* erzeugt. Mit Hilfe dieses Anweisungsobjekts erfolgen alle Abfragen und Veränderungen der Datenbank.

Falls die gewünschte Datenbank nicht geöffnet werden konnte, löst die Methode *getConnection()* eine Ausnahme des Typs *SQLException* aus. Diese muss in einer *catch*-Anweisung behandelt werden.

Am Ende einer Datenbanksitzung wird die Verbindung zur Datenbank in der Methode *schliesseVerbindung()* wieder beendet. Dazu werden das *Statement*-Objekt und das *Connection*-Objekt jeweils mit der Methode *close()* explizit geschlossen. Die Verbindung wird automatisch beendet, wenn das *Connection*-Objekt vom Garbage Collector zerstört wird.

```
public void schliesseVerbindung()
{
    try {
        stmt.close();
        con.close();
    }
    catch (SQLException sqle) {
        System.out.println(sqle.toString());
    }
}
```

Abbildung 3.3: Die Methode *schliesseVerbindung()* in der Klasse *DBVerbindung*

Übung 3.1:

Erstelle eine Kopie der Datenbank *Kaffeeladen1* unter dem Namen *Kaffeehaus1*

Erstelle in **BlueJ** ein Projekt *Kaffee01a*. Implementiere in der Klasse *DBVerbindung*

1. die Datenfelder und den Konstruktor (Abbildung 3.1),
2. die Methode *erstelleVerbindung()* (Abbildung 3.2) und
3. die Methode *schliesseVerbindung* (Abbildung 3.3).

Nun kann ein Objekt der Klasse *DBVerbindung* erzeugt werden und diesem die benötigten Parameterwerte übergeben werden. Abbildung 3.4 zeigt das entsprechende Dialogfeld in **BlueJ**.



Abbildung 3.4: Erstellen des Objekts *dbVerbindung1*

Übung 3.2:

Erzeuge in **BlueJ** ein Objekt der Klasse *DBVerbindung*. Untersuche, ob die beiden Methoden *erstelleVerbindung()* und *schliesseVerbindung()* ohne Fehlermeldung aufgerufen werden können. Diese Klasse hat bis noch keine größere Funktionalität-

Du kannst nun die Verbindung zu der MySQL-Datenbank *Kaffeehaus1* herstellen und diese auch wieder beenden. Nun wirst du lernen, wie man eine einfache Datenbankabfrage durchführt. Zu jedem Kaffee sollen seine gesamten Daten aufgelistet werden, also *KaffeeID*, *Name*, *Anbieter*, *Sorte*, *Preis* und *Anzahl*.

Zum Beispiel liefert die SQL-Anweisung an die Datenbank *Kaffeehaus1*

```
SELECT * FROM Kaffee
```

aus der Tabelle *Kaffee* die Werte aller Datenfelder.

In Java wird dazu das *Statement*-Objekt *stmt* verwendet, um mit dessen Methode *executeQuery()* Daten aus der Datenbank zu lesen. Die Methode *executeQuery()* erwartet einen SQL-String in Form einer gültigen SELECT-Anweisung und gibt ein Objekt *results* vom Typ *ResultSet* zurück, das die Ergebnistabelle repräsentiert.

```
ResultSet results = stmt.executeQuery("SELECT * FROM Kaffee");
```

Die Variable *results* enthält nun in ihren Zeilen alle gewünschten Daten von jedem Kaffee, der in dieser Tabelle gespeichert wurde.

KaffeeID	Name	Sorte	Preis	Anzahl	Anbieter
1	Espresso dOro	Espresso	7.30	0	Dallmayr
2	Brasil Mild	Bohnenkaffee	6.40	50	Tchibo
3	Brasilian	Pulverkaffee	4.80	0	Dallmayr
usw.					

Abbildung 3.5: Das Objekt *results* repräsentiert die Ergebnistabelle der Abfrage „SELECT * FROM Kaffee“

Bemerkung:

Die Großschreibung der Schlüsselwörter in den SQL-Anweisungen ist nicht notwendig. Sie dient ausschließlich der besseren Lesbarkeit und zur Unterscheidung zwischen Schlüsselwörter und benutzerdefinierten Bezeichnern.

Das *ResultSet*-Objekt *results* besitzt eine Art Cursor, der auf eine entsprechende Zeile der Tabelle zeigt. Mit Hilfe der Methode *next()* kann man nun zeilenweise durch diese Tabelle wandern. Dazu wird eine *while*-Schleife verwendet. Mit Hilfe der Methoden *getString()*, *getInt()* und *getDouble()* können nun aus jeder Zeile die entsprechenden Werte herausgelesen und z.B. im Terminalfenster aufgelistet werden

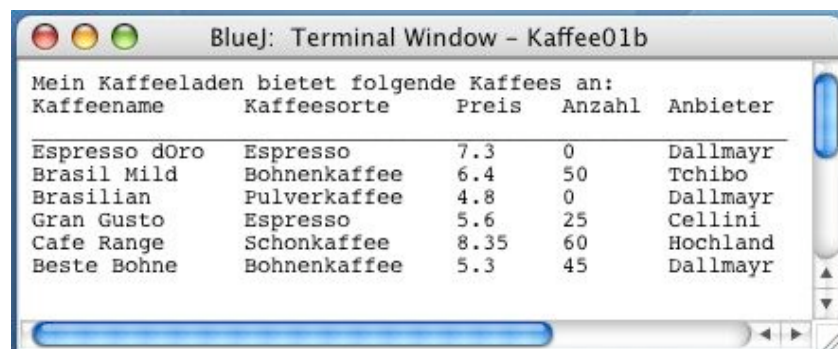
```
/** Zeigt Inhalte einiger Datenfelder aus der gewünschten Tabelle. */
public void holeDaten()
{
    try {
        ResultSet results = stmt.executeQuery("SELECT * FROM Kaffee");
        System.out.println("Mein Kaffeeladen bietet folgende Kaffees an:");
        System.out.println("Kaffeename\tKaffeessorte\tPreis\tAnzahl\tAnbieter");
        System.out.println("-----");
        while (results.next()) {
            //int kaffeeID = results.getInt("KaffeeID");
            String name = results.getString("Name");
            String sorte = results.getString("Sorte");
            double preis = results.getDouble("Preis");
            int anzahl = results.getInt("Anzahl");
            String anbieter = results.getString("Anbieter");
            System.out.println(name + "\t" + sorte + "\t" +
                preis + "\t" + anzahl + "\t" + anbieter);
        }
        System.out.println();
    }
    catch (SQLException sqle) {
```

```
        System.out.println(sql.toString());
    }
}
```

Abbildung 3.6: Die Methode *holeDaten()* in der Klasse *DBVerbindung*

Übung 3.3:

Speichere dein Projekt nun unter dem Namen *Kaffee01b*. Implementiere die Methode *holeDaten()* in der Klasse *DBVerbindung*. Überprüfe dein Ergebnis mit der Datenbank *Kaffeehaus1* (in phpMyAdmin). Abbildung 3.7 zeigt die Ausgabe im Terminalfenster.



Mein Kaffeeladen bietet folgende Kaffees an:				
Kaffeename	Kaffeesorte	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.3	0	Dallmayr
Brasil Mild	Bohnenkaffee	6.4	50	Tchibo
Brasilian	Pulverkaffee	4.8	0	Dallmayr
Gran Gusto	Espresso	5.6	25	Cellini
Cafe Range	Schonkaffee	8.35	60	Hochland
Beste Bohne	Bohnenkaffee	5.3	45	Dallmayr

Abbildung 3.7: Terminalfenster zeigt das Ergebnis der Abfrage

Bemerkungen:

1. Die Preis wird wegen des Datentyps *double* nicht mit zwei Dezimalstellen dargestellt, falls die letzte Ziffer eine 0 ist.
2. JDBC bietet eine weitere Möglichkeit, mit Hilfe der Methoden *getXXX()*, die entsprechenden Werte aus einer Zeile zu erhalten. Man kann anstelle des Spaltennamens auch den Spaltenindex (Nummer der Spalte) verwenden:

```
String name = results.getString(1);
double preis = results.getDouble(4);
```

Es erscheint allerdings nicht sehr sinnvoll, in der Klasse *DBVerbindung* die Datenbankabfragen zu implementieren. Diese Abfragen werden in einer separaten Klasse *DBAbfrage* behandelt und sollten auch hier implementiert werden.

```
public class DBAbfrage
{
    private DBVerbindung dbVerbindung;
```

```
public DBAbfrage()
{
    dbVerbindung = new DBVerbindung("Kaffeehaus", "ralph", "ralph");
}

/** Zeigt Inhalte einiger Datenfelder aus der gewuenschten Tabelle. */
public void holeDaten()
{
    String qryAlleDaten = "SELECT * " +
        "FROM Kaffee";

    dbVerbindung.erstelleVerbindung();
    Statement stmt = dbVerbindung.gibStmt();

    try {
        ResultSet results = stmt.executeQuery(qryAlleDaten);

        ..... weitere Anweisungen in try und catch .....
    }

    dbVerbindung.schliesseVerbindung();
}
}
```

Abbildung 3.8: Quelltext der Klasse *DBAbfrage* mit der Methode *holeDaten()*

Abbildung 3.8 zeigt, dass im Konstruktor ein Objekt *dbVerbindung* mit den bereits bekannten Parametern erzeugt wird. Die Methode *holeDaten()* stellt zuerst die Datenbankverbindung her und holt sich anschließend das *Statement*-Objekt *stmt*, Dazu muss allerdings in der Klasse *DBVerbindung* zusätzlich eine Methode *gibStmt()* implementiert werden.

```
/** Liefert das Anweisungsobjekt zu Abfragen und Aenderungen der DB. */
public Statement gibStmt()
{
    return stmt;
}
```

Abbildung 3.9: Methode *gibStmt()* in der Klasse *DBVerbindung*

Nun folgen die aus *Kaffee01b* bekannten Anweisungen der Methode *holeDaten()*. Ich habe zusätzlich die SQL-Abfrage als String *qryAlleDaten* gespeichert und der Methode *executeQuery()* als Parameter übergeben. Zum Schluss muss noch die Verbindung zur Datenbank wieder unterbrochen werden.

Übung 3.4:

Speichere dein Projekt unter dem Namen *Kaffee02*. Erstelle eine neue Klasse *DBAbfrage* und kopiere die Methode *holeDaten()* mit den oben erwähnten Änderungen in diese Klasse.

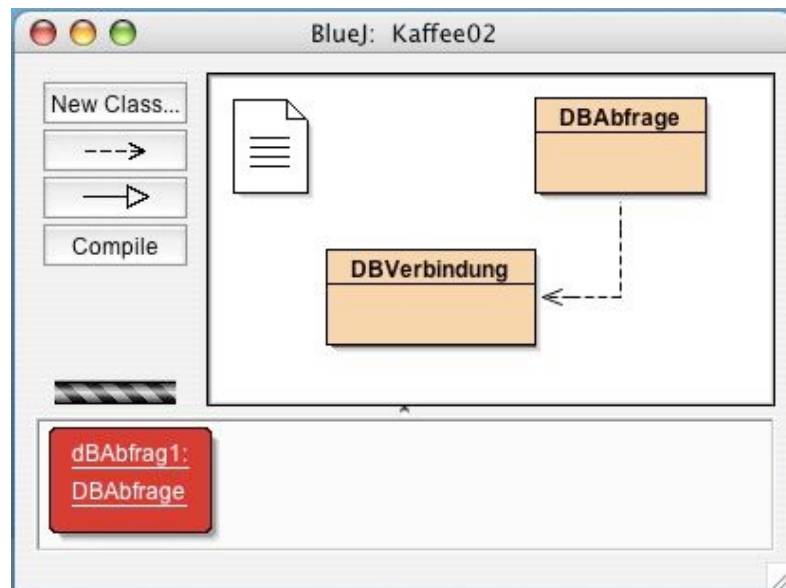


Abbildung 3.10: Klassendiagramm des Projekts *Kaffee02*

Kaffeename	Kaffeesor	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.3	0	Dallmayr
Brasil Mild	Bohnenkaffee	6.4	50	Tchibo
Brasilian	Pulverkaffee	4.8	0	Dallmayr
Gran Gusto	Espresso	5.6	25	Cellini
Cafe Range	Schonkaffee	8.35	60	Hochland
Beste Bohne	Bohnenkaffee	5.3	45	Dallmayr

Abbildung 3.11: Terminalfenster zeigt das Ergebnis der Abfrage

Du wirst in den folgenden Beispielen die gesamten Daten eines bestimmten Kaffees anzeigen lassen. Das Herauslesen der Daten aus dem *results* erfolgt jedoch immer auf die gleiche Weise. Deshalb solltest du diese Aufgabe in eine separate Methode *befrageResults()* legen.

```
/** Holt sich die Werte aus results und druckt diese aus. */
private void befrageResults(ResultSet results)
{
    try {
        System.out.println("Mein Kaffeeladen bietet folgende Kaffees an:");
        System.out.println("Kaffeename\tKaffeesor\tPreis\tAnzahl\tAnbieter");
    }
}
```

```
System.out.println("-----");
while (results.next()) {
    //int kaffeeID = results.getInt("KaffeeID");
    String name = results.getString("Name");
    String sorte = results.getString("Sorte");
    double preis = results.getDouble("Preis");
    int anzahl = results.getInt("Anzahl");
    String anbieter = results.getString("Anbieter");
    System.out.println(name + "\t" + sorte + "\t" +
        preis + "\t" + anzahl + "\t" + anbieter);
}
System.out.println();
}
catch (SQLException sqle) {
    System.out.println(sqle.toString());
}
}
```

Abbildung 3.12: Die Methode *befrageResults()* in der Klasse *DBAbfrage*

Übung 3.5:

Implementiere die Methode *befrageResults()* in der Klasse *DBAbfrage* deines Projekt *Kaffee02*. Überprüfe nun, ob dein Projekt das korrekte Ergebnis liefert.

Bemerkung:

Dieses Projekt *Kaffee02* dient als Vorlage für alle weiteren Projekte in diesem Kapitel, um die Grundlagen von Datenbankzugriffen zu erläutern. Von diesem Projekt wirst du immer wieder Kopien anlegen müssen.

3.1.1 Projektion

Nun wirst Du eine einfache Abfrage an die Datenbank *Kaffeehaus1* stellen

Übung 3.6:

Speichere das Projekt *Kaffee02* unter dem Namen *Kaffee02a*. Erstelle in diesem Projekt nun folgende Abfragen über die Projektion:

Gesucht sind die Daten aller Kaffees. Dabei ist die Ergebnistabelle absteigend nach Anbieter sortiert auszugeben. Implementiere dazu die Methode *holeAlleKaffeesSortiert()*. Vergleiche dazu auch die sortierte Ausgabe der Kaffees in Abbildung 3.14.

Mein Kaffeeladen bietet folgende Kaffees an:				
Kaffeename	Kaffeesor	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.3	0	Dallmayr
Brasil Mild	Bohnenkaffee	6.4	50	Tchibo
Brasilian	Pulverkaffee	4.8	0	Dallmayr
Gran Gusto	Espresso	5.6	25	Cellini
Cafe Range	Schonkaffee	8.35	60	Hochland
Beste Bohne	Bohnenkaffee	5.3	45	Dallmayr

Mein Kaffeeladen bietet folgende Kaffees an:				
Kaffeename	Kaffeesor	Preis	Anzahl	Anbieter
Cafe Range	Schonkaffee	8.35	60	Hochland
Espresso dOro	Espresso	7.3	0	Dallmayr
Brasil Mild	Bohnenkaffee	6.4	50	Tchibo
Gran Gusto	Espresso	5.6	25	Cellini
Beste Bohne	Bohnenkaffee	5.3	45	Dallmayr
Brasilian	Pulverkaffee	4.8	0	Dallmayr

Abbildung 3.13: Die Kaffees werden nach Preis in absteigender Reihenfolge sortiert.

Die zugehörige SQL-Anweisung lautet:

```
String qryAlleDaten = "SELECT * " +
    "FROM Kaffee " +
    "ORDER BY Preis DESC";
```

Übung 3.7:

- Gesucht sind nur der Name und der Preis aller Kaffees sortiert nach Preis.
- Gesucht sind nur der Name und die Sorte aller Kaffees sortiert nach Sorte.

3.1.2 Selektion

Nun wirst Du aus der Datenbank *Kaffeehaus1* nur die Daten über den Kaffee „Brasilian“ abfragen. Hierzu wird wiederum das *Statement*-Objekt *stmt* verwendet, um mit dessen Methode *executeQuery()* Daten aus der Datenbank zu lesen.

Die Methode *executeQuery()* erwartet einen SQL-String in Form einer gültigen SELECT-Anweisung und gibt ein Objekt *results* vom Typ *ResultSet* zurück, das die Ergebnismenge repräsentiert.

In dieser Abfrage liefert die SQL-Anweisung an die Datenbank *Kaffeehaus1*

```
SELECT *
FROM Kaffee
WHERE Name LIKE 'Brasilian'
```

aus der Tabelle *Kaffee* alle Daten ausschließlich vom Kaffee „Brasilian“.

```
/** Zeigt die Daten des Kaffees Brazilian*/
public void holeBrazilian()
{
    String qryBrazilian = "SELECT * " +
                          "FROM Kaffee " +
                          "WHERE Name LIKE 'Brazilian'";

    dbVerbindung.erstelleVerbindung();
    Statement stmt = dbVerbindung.gibStmt();

    try {
        ResultSet results = stmt.executeQuery(qryBrazilian);

        ..... weitere Anweisungen in try und catch .....
    }

    dbVerbindung.schliesseVerbindung();
}
```

Abbildung 3.14: Die Methode *holeBrazilian()* in der Klasse *DBAbfrage*

Bemerkung:

Beachte die einzelnen Anführungsstriche um den Namen, da dieser Name in den doppelten Anführungszeichen der SQL-Anweisung enthalten ist.

Übung 3.8:

Speichere das Projekt *Kaffee02* unter dem Namen *Kaffee02b*. Erstelle in diesem Projekt nun folgende Abfragen über die Selektion:

- Gesucht sind alle Daten über die Kaffeessorte „Brasilian“. Implementiere dazu die in Abbildung 3.14 dargestellte Methode *holeBrazilian()*.
- Gesucht sind alle Kaffeessorten, die ausverkauft sind
- Gesucht sind alle Kaffeessorten, die weniger als 6.90 € kosten.
-

In den bisherigen Beispielen waren die Bedingungen fest vorgegeben. Interessant ist nun die Verwendung von Parametern für diese Bedingungen. Im folgenden Beispiel fragt der Benutzer nach allen Daten eines Kaffees, dessen Namen als Parameterwert übergeben wird. Die Methode *executeQuery()* erwartet einen SQL-String in der Form

```
SELECT *
FROM sorte
```

WHERE Name LIKE <kaffeename>

wobei der Parameter *kaffeename* den vom Benutzer gewünschten Kaffee enthält.

Der Methode *executeQuery()* wird also nun folgender String *qryKaffee* übergeben:

```
String qryName = "SELECT * " +  
                "FROM Kaffee " +  
                "WHERE Name LIKE '" + kaffeename + "'";
```

Bemerkung:

Krüger schachtelt auf S. 972 im Listing 42.7 den Parameter *kaffeename* in einfache Anführungszeichen mit BackSlash ein:

```
String qryName = "SELECT * " +  
                "FROM Kaffee " +  
                "WHERE Name LIKE \'\' + kaffeename + \'\'";
```

Ich habe noch keine Erklärung gefunden, warum es diesen BackSlash verwendet. Auf S.91 listet er die Sonderbedeutungen mit BackSlash auf.

Übung 3.9:

- Erstelle im Projekt *Kaffee02b* eine Methode *holeKaffeeName()*, die den Benutzer fragt, von welchem Kaffee er die Daten wissen will. Diese könnte als Parameter z.B. den String *kaffeename* verwenden. Orientiere dich an der Methode *holeBrasilian()*. Vergleiche dazu das Dialogfenster in Abbildung 3.15.
- Gesucht sind alle Kaffees, die weniger/mehr als einen bestimmten Betrag kosten.
- Gesucht sind alle Kaffees, die zu einer bestimmten Kaffeesorste gehören.



Abbildung 3.15: Dialogfenster zur Parametereingabe

Bemerkung:

Weitere wichtige Hinweise zu der SQL-Anweisung `SELECT` findest du in *MySQL, Das Einsteigerseminar* auf den Seiten 127 bis 136.

3.2 Datenbankänderungen

3.2.1 Ändern von Datensätzen

Du willst nun als Besitzer des Kaffeeladens nach einer erfolgreichen Woche in deiner Datenbank *Kaffeehaus1* die Anzahl der Packungen, die noch in den Regalen stehen, eintragen. Das bedeutet, dass du nun bereits bestehende Datensätze ändern muss.

Dazu wird das *Statement*-Objekt *stmt* verwendet, um mit dessen Methode *executeUpdate()* Daten in der Datenbank zu ändern. Die Methode *executeUpdate()* erwartet einen SQL-String in Form einer gültigen UPDATE-Anweisung.

Zum Beispiel ändert die SQL-Anweisung an der Datenbank *Kaffeeladen*

```
UPDATE Kaffee
SET Anzahl = 75
WHERE Name LIKE 'Espresso dOro'
```

in der Tabelle *Kaffee* den entsprechenden Datensatz ein.

Bemerkung:

Beachte die einzelnen Anführungsstriche um den Namen, da dieser Name in den doppelten Anführungszeichen der SQL-Anweisung enthalten ist.

```
/** Aendert vom Kaffee Espresso dOro das Datenfeld Anzahl. */
public void aendereAnzahlOro()
{
    String updOro = "UPDATE Kaffee " +
                   "SET Anzahl = 75 " +
                   "WHERE Name LIKE 'Espresso dOro'";

    dbVerbindung.erstelleVerbindung();
    Statement stmt = dbVerbindung.gibStmt();

    try {
        stmt.executeUpdate(updOro);
    }
    catch (SQLException sqle) {
```

```

        System.out.println(sql.toString());
    }

    dbVerbindung.schliesseVerbindung();
}

```

Abbildung 3.16: Die Methode *aenderAnzahlWColombian()*

Übung 3.10:

Speichere das Projekt *Kaffee02* unter dem Namen *Kaffee02c*. Implementiere die in Abbildung 3.16 dargestellte Methode *aendereAnzahlOro()*. Rufe nacheinander die Methoden *holeAlleKaffees()*, *aendereAnzahlOro()* und *holeAlleKaffees()* auf.

- Vergleiche dein Ergebnis mit dem in Abbildung 3.17 dargestellten Terminal-Fenster.
- Überprüfe auch die Datenbank *Kaffeehaus1* in *phpMyAdmin*.



Kaffeeiname	Kaffeesorste	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.3	75	Dallmayr
Brasil Mild	Bohnenkaffee	6.4	50	Ichibo
Brasilian	Pulverkaffee	4.8	0	Dallmayr
Gran Gusto	Espresso	5.6	25	Cellini
Cafe Range	Schonkaffee	8.35	60	Hochland
Beste Bohne	Bohnenkaffee	5.3	45	Dallmayr

Abbildung 3.17: Die Anzahl von Kaffee *Espresso dOro* wurde geändert

Übung 3.11:

- Der Kaffee „Classico“ findet reißenden Absatz fand. Du erhöhst deshalb den Preis auf 6.70 Euro.
- Der Kaffee „Brasil Mild“ wird zum Ladenhüter. Du erniedrigst deshalb den Preis auf 5.90 Euro.
- Du hast festgestellt, dass der Kaffee „Mocca Gold“ ein Schonkaffee ist. Verbessere deine Datenbank.

Bemerkung:

Weitere wichtige Hinweise zu der SQL-Anweisung UPDATE findest du in *MySQL, Das Einsteigerseminar* auf den Seiten 140ff

3.2.2 Einfügen von Datensätzen

Nun wirst Du in die Tabelle *Kaffee* der Datenbank *Kaffeehaus1* einen neuen Datensatz einfügen. Im Sortiment hast du nun den neuen Kaffee mit

Namen	Feine Milde
Anbieter	Hochland
Sorte	Schonkaffee
Preis	7.25

aufgenommen.

Dazu wird das *Statement*-Objekt *stmt* verwendet, um mit dessen Methode *executeUpdate()* Daten in die Datenbank einzulesen. Die Methode *executeUpdate()* erwartet einen SQL-String in Form einer gültigen INSERT-Anweisung.

Die Eingabe der SQL-Anweisung

```
INSERT INTO Kaffee
VALUES ('Feine Milde', 'Hochland', 'Schonkaffee', 7.25)
```

führt jedoch zu folgender Fehlermeldung:

java.sql.SQLException: Column count doesn't match value count at row 1,
denn die Tabelle *Kaffee* besteht aus sechs Datenfeldern mit zum Teil
voreingestellten Werten:

KaffeeID		auto-increment
Namen	Feine Milde	
Anbieter	Hochland	
Sorte	Schonkaffee	
Preis	7.25	
Anzahl		0

Nur wenn du genauso viele Werte übergibst, wie die Tabelle Datenfelder besitzt, darfst du die Attributliste in der SQL-Anweisung weglassen.

In dieser Tabelle *Kaffee* gilt, dass du *KaffeeID* mit einer *auto_increment*-Option versehen hast und für *Anzahl* den DEFAULT-Wert 0 angegeben hast. Deshalb musst du alle Datenfelder außer *KaffeeID* und *Anzahl* in der Attributliste angeben.

```
INSERT INTO Kaffee(Name, Anbieter, Sorte, Preis)
VALUES ('Feine Milde', 'Hochland', 'Schonkaffee', 7.25)
```

In diesem Fall vergibt die Datenbank automatisch die korrekte *KaffeeID* und setzt den Wert von *Anzahl* auf 0.

Bemerkung:

- a) Beachte die einzelnen Anführungsstriche um den Namen, da dieser Name in den doppelten Anführungszeichen der SQL-Anweisung enthalten ist.
- b) Weitere wichtige Hinweise zu der SQL-Anweisung INSERT findest du in *MySQL, Das Einsteigerseminar* auf den Seiten 125f.

```
/** Fuegt einen weiteren Datensatz in die Tabelle Kaffee. */
public void neuMilde()
{
    String updMilde = "INSERT INTO Kaffee(Name, Anbieter, Sorte, Preis) " +
        "VALUES ('Feine Milde', 'Hochland', 'Schonkaffee', 7.25)";

    dbVerbindung.erstelleVerbindung();
    Statement stmt = dbVerbindung.gibStmt();

    try {
        stmt.executeUpdate(updMilde);
    }
    catch (SQLException sqle) {
        System.out.println(sqle.toString());
    }

    dbVerbindung.schliesseVerbindung();
}
```

Abbildung 3.18: Die Methode *neuColombina()*

Übung 3.12:

Speichere das Projekt *Kaffee02* unter dem Namen *Kaffee02d*. Implementiere die in Abbildung 2.16 dargestellte Methode *neuMilde()*. Rufe nacheinander die Methoden *holeAlleKaffees()*, *neuMilde()*, und *holeAlleKaffees()* auf.

- a) Überprüfe den neuen Datensatz im Terminalfenster.
- b) Überprüfe auch die Datenbank *Kaffeehaus1* in *phpMyAdmin*.
- c) Rufe noch einmal die beiden Methoden *neuMilde()* und *holeAlleKaffees()* auf. Warum steht scheinbar der gleiche Datensatz ein zweites Mal in der Tabelle *Kaffee*?

Übung 3.13:

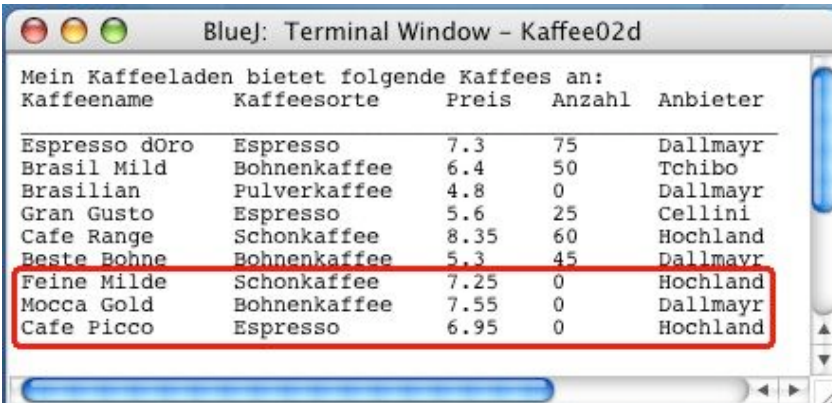
Implementiere in der Klasse *DBAbfrage* eine Methode *neuMoccaPicco()*, die gleichzeitig die beiden Datensätze

```
('Mocca Gold', 'Dallmayr', 'Bohnenkaffee' 7.55),
('Cafe Picco', 'Hochland', 'Espresso', 6.95)
```

in die Tabelle *Kaffee* einfügt.

- Überprüfe deine Veränderung im Terminal-Fenster.
- Überprüfe auch die Datenbank *Kaffeehaus1* in *phpMyAdmin*

Falls der Kaffee „Feine Milde“ immer noch zweimal in deiner Tabelle *Kaffee* vorhanden ist, lösche einen davon mit Hilfe von *phpMyAdmin*!



Kaffeename	Kaffeesorste	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.3	75	Dallmayr
Brasil Mild	Bohnenkaffee	6.4	50	Tchibo
Brasilian	Pulverkaffee	4.8	0	Dallmayr
Gran Gusto	Espresso	5.6	25	Cellini
Cafe Range	Schonkaffee	8.35	60	Hochland
Beste Bohne	Bohnenkaffee	5.3	45	Dallmayr
Feine Milde	Schonkaffee	7.25	0	Hochland
Mocca Gold	Bohnenkaffee	7.55	0	Dallmayr
Cafe Picco	Espresso	6.95	0	Hochland

Abbildung 3.19: Die neuen Kaffees wurden eingefügt

3.2.3 Verwendung von Prepared Statements (für Experten)

*PreparedStatement*s sind parametrisierte SQL-Anweisungen. Sie werden zunächst deklariert und zum Vorkompilieren an die Datenbank übergeben. Später können sie dann beliebig oft ausgeführt werden, indem die formalen Parameter durch aktuelle Werte ersetzt werden. In diesem ersten Beispiel wirst du die Anzahl der Packungen des Kaffees Colombian auf 65 setzen.

Zuerst muss man sich zusätzlich zum Anweisungsobjekt *stmt* auch noch das Verbindungsobjekt *con* aus der Klasse *DBVerbindung* holen. In dieser Klasse wird also noch eine weitere Methode *gibCon()* implementiert.

```
/** Liefert das Verbindungsobjekt zur DB. */
public Connection gibCon()
{
    return con;
}
```

Abbildung 3.20: Die Methode *gibCon()* in der Klasse *DBVerbindung*

Dieses Verbindungsobjekt *con* liefert mit Hilfe seiner Methode *prepareStatement()* das vorbereitete Anweisungsobjekt *updAnzahl*. Der Methode *prepareStatement()* muss als Argument ein String übergeben werden, der die gewünschte SQL-Anweisung enthält. In diesem werden die formalen Parameter durch Fragezeichen dargestellt.

```
UPDATE Kaffee
SET Anzahl = ?
WHERE Name LIKE ?
```

In dieser parametrisierten SQL-Anweisung wird in der Tabelle *Kaffee* dem Datenfeld *Anzahl* ein neuer Wert (erstes Fragezeichen) zugewiesen. Das zweite Fragezeichen erwartet den Namen desjenigen Kaffees, dessen *Anzahl* geändert werden soll.

Mit Hilfe der Methoden *setXXX()* kann man nun den Wert des Datenfeldes *Anzahl* auf den Wert 65 setzen beim Kaffee mit dem Name „Feine Milde“. Zum Schluss wird diese Änderung mit *executeUpdate()* ausgeführt und mit *close()* wieder geschlossen.

```
/** Aendert die Datensaeetze mit Hilfe eines PreparedStatement. */
public void aendereDatensatz()
{
    String updString = "UPDATE Kaffee " +
        "SET Anzahl = ? " +
        "WHERE Name Like ? ";

    dbVerbindung.erstelleVerbindung();
    Statement stmt = dbVerbindung.gibStmt();
    Connection con = dbVerbindung.gibCon();

    try {
        PreparedStatement updAnzahl = con.prepareStatement(updString);
        //aendert Anzahl von Feine Milde auf 75
        updAnzahl.setInt(1, 65);
        updAnzahl.setString(2, "Feine Milde");
        updAnzahl.executeUpdate();

        updAnzahl.close();
    }
    catch (SQLException sqle) {
        System.out.println(sqle.toString());
    }
}
```

```
    dbVerbindung.schliesseVerbindung();  
}
```

Abbildung 3.21: Der Kaffee „Feine Milde“ ist noch 65 mal vorhanden.

Übung 3.14:

Speichere das Projekt *Kaffee02* unter dem Namen *Kaffee02e*. Implementiere in der Klasse *DBAbfrage* die Methode *gibCon()* und in der Klasse *DBAbfrage* die Methode *aendereDatensatz()*.

Analog lässt sich auch jeweils die Anzahl der Kaffees „Mocca Gold“ und „Cafe Picco“ verändern.

```
//aendert Anzahl von Mocca Gold auf 100  
updAnzahl.setInt(1, 100);  
updAnzahl.setString(2, "Mocca Gold");  
updAnzahl.executeUpdate();  
//aendert Anzahl von Cafe Picco auf 100  
//erster Parameter bleibt 100  
//zweiter Parameter wird auf Cafe Picco gesetzt  
updAnzahl.setString(2, "Cafe Picco");  
updAnzahl.executeUpdate();
```

Abbildung 3.22: Weitere Zahlen in der Methode *aendereDatensatz()*

Übung 3.15:

Vervollständige die Methode *aendereDatensatz()* und überprüfe, ob die Anzahl der noch vorhandenen Packungen in der Datenbank *Kaffeehaus1* geändert wurden. Vergleiche dazu *Abbildung 3.21*.



Bluej: Terminal Window - Kaffee02e

Mein Kaffeeladen bietet folgende Kaffees an:

Kaffeename	Kaffeesor	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.3	75	Dallmayr
Brasil Mild	Bohnenkaffee	6.4	50	Tchibo
Brasilian	Pulverkaffee	4.8	0	Dallmayr
Gran Gusto	Espresso	5.6	25	Cellini
Cafe Range	Schonkaffee	8.35	60	Hochland
Beste Bohne	Bohnenkaffee	5.3	45	Dallmayr
Feine Milde	Schonkaffee	7.25	65	Hochland
Mocca Gold	Bohnenkaffee	7.55	100	Dallmayr
Cafe Picco	Espresso	6.95	100	Hochland

Abbildung 3.23: Die Anzahl weiterer Kaffees wurde geändert

Der Quelltext in Abbildung 3.22 zeigt, dass es sehr umständlich ist, alle wöchentlichen Verkaufszahlen in einer umfangreichen Tabelle zu aktualisieren. Hier führt die Verwendung einer Schleife zu einer erheblichen Vereinfachung. Allerdings müssen die aktuellen Verpackungszahlen und die Kaffeennamen in den Arrays *anzahl* und *namen* verwaltet werden.

```
/** Aendert die Datensaeetze mit Hilfe einer Schleife. */
public void aendereRestlicheDatensaeetze()
{
    String updString = "UPDATE Kaffee " +
        "SET Anzahl = ? " +
        "WHERE Name Like ? ";

    dbVerbindung.erstelleVerbindung();
    Statement stmt = dbVerbindung.gibStmt();
    Connection con = dbVerbindung.gibCon();

    try {
        PreparedStatement updAnzahl = con.prepareStatement(updString);

        int[] anzahl = {175, 150, 160, 155, 190};
        //Alternative: macht die Aenderungen wieder rueckgaengig
        //int[] anzahl = {50, 0, 25, 60, 45};
        String[] namen = {"Brasil Mild", "Brasilian", "Gran Gusto",
            "Cafe Range", "Beste Bohne"};

        int laenge = anzahl.length;
        for (int i = 0; i < laenge; i++) {
            updAnzahl.setInt(1, anzahl [i]);
            updAnzahl.setString(2, namen[i]);
            updAnzahl.executeUpdate();
        }

        updAnzahl.close();
    }
    catch (SQLException sqle) {
        System.out.println(sqle.toString());
    }

    dbVerbindung.schliesseVerbindung();
}
```

Abbildung 3.24: Verwendung einer Schleife

Übung 3.16:

Implementiere die in Abbildung 2.26 dargestellte Methode *aendereRestlicheDatensatz()*. Vergleiche dazu Abbildung 2.27!

Kaffeename	Kaffeesorste	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.3	75	Dallmayr
Brasil Mild	Bohnenkaffee	6.4	175	Tchibo
Brazilian	Pulverkaffee	4.8	150	Dallmayr
Gran Gusto	Espresso	5.6	160	Cellini
Cafe Range	Schonkaffee	8.35	155	Hochland
Beste Bohne	Bohnenkaffee	5.3	190	Dallmayr
feine Milde	Schonkaffee	7.25	65	Hochland
Mocca Gold	Bohnenkaffee	7.55	100	Dallmayr
Cafe Picco	Espresso	6.95	100	Hochland

Abbildung 3.25: Die Methode *aendereRestlicheDatensaetze()* aktualisiert die restlichen Datensätze

3.3 Verwendung von Transaktionen

In den bisherigen Beispielen war es nicht möglich, bei einem Datensatz gleichzeitig mehrere Veränderungen durchzuführen. Nach dem Erzeugen einer Verbindung zur Datenbank, befindet sich diese (gemäß JDBC-Spezifikationen) normalerweise im *auto-commit*-Modus, das bedeutet dass jede einzelne SQL-Anweisung als separate Transaktion angesehen wird, die nach dem Ende des Kommandos automatisch bestätigt wird.

Diesen *auto-commit*-Modus mit Hilfe der Methode *setAutoCommit(false)* zu deaktivieren stellt eine Möglichkeit dar, zwei oder mehr SQL-Anweisungen in einer Transaktion zu gruppieren. All diese Anweisungen müssen dann allerdings explizit durch die Methode *commit()* bestätigt werden. Anschließend muss der *auto-commit*-Modus mit *setAutoCommit(true)* wieder aktiviert werden.

Eine Transaktion wirst du mit Hilfe des folgenden Beispiels durchführen: Am Ende eines weiteren erfolgreichen Tages hast du in 45 Packungen des Kaffees „Brasil Mild“ verkauft und gleichzeitig willst du diesen Kaffee um 5% verteuern.

```
/** Aendert Preis und gleichzeitig Anzahl von Brasil Mild. */
public void transBrasil()
{
    String updPreis = "UPDATE Kaffee " +
        "SET Preis = Preis * 1.05 " +
        "WHERE Name LIKE 'Brasil Mild'";
    String updAnzahl = "UPDATE Kaffee " +
        "SET Anzahl = Anzahl - 45 " +
        "WHERE Name LIKE 'Brasil Mild'";
```

```

dbVerbindung.erstelleVerbindung();
Statement stmt = dbVerbindung.gibStmt();
Connection con = dbVerbindung.gibCon();

try {
    con.setAutoCommit(false);
    stmt.executeUpdate(updPreis);
    stmt.executeUpdate(updAnzahl);
    con.commit();
    con.setAutoCommit(true);
}
catch (SQLException sqle) {
    System.out.println(sqle.toString());
}

dbVerbindung.schliesseVerbindung();
}

```

Abbildung 3.26: Die Methode *transBrasil()* in der Klasse *DBAbfrage*

Übung 3.17:

Speichere das Projekt *Kaffee02* unter dem Namen *Kaffee02g*. Implementiere die in Abbildung 3.26 dargestellte Methode *transBrasil()*.
Vergleiche die Abbildung 3.27.



Bluej: Terminal Window - Kaffee02f

Mein Kaffeeladen bietet folgende Kaffees an:

Kaffeename	Kaffeesor	Preis	Anzahl	Anbieter
Espresso d'oro	Espresso	7.3	75	Dallmayr
Brasil Mild	Bohnenkaffee	6.72	5	Tchibo
Brasilian	Pulverkaffee	4.8	0	Dallmayr
Gran Gusto	Espresso	5.6	25	Cellini
Cafe Range	Schonkaffee	8.35	60	Hochland
Beste Bohne	Bohnenkaffee	5.3	45	Dallmayr
Feine Milde	Schonkaffee	7.25	65	Hochland
Mocca Gold	Bohnenkaffee	7.55	100	Dallmayr
Cafe Picco	Espresso	6.95	100	Hochland

Abbildung 3.27: Anzahl um 55 verringert und Preis um 5% erhöht

Übung 2.19:

- Implementiere eine Transaktions-Methode *transBrazilian()*, die die Attribute *Preis* und *Anzahl* der Kaffeesor „Brasilian“ mit Hilfe von *PreparedStatement* verändert.
- Für Experten:

Implementiere eine Transaktions-Methode *transRestlicheKaffees()*, die die Attribute *Preis* und *Anzahl* bei allen restlichen Kaffees mit Hilfe von *PreparedStatement*s verändert.

The screenshot shows a terminal window titled "Bluej: Terminal Window - Kaffee02f". It displays two tables of coffee data. The first table shows the initial state, and the second table shows the state after the *transRestlicheKaffees()* method has been executed, with updated prices and quantities.

Mein Kaffeeladen bietet folgende Kaffees an:				
Kaffeename	Kaffeessorte	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.3	75	Dallmayr
Brasil Mild	Bohnenkaffee	6.72	5	Tchibo
Brasilian	Pulverkaffee	5.04	60	Dallmayr
Gran Gusto	Espresso	5.6	25	Cellini
Cafe Range	Schonkaffee	8.35	60	Hochland
Beste Bohne	Bohnenkaffee	5.3	45	Dallmayr
Feine Milde	Schonkaffee	7.25	65	Hochland
Mocca Gold	Bohnenkaffee	7.55	100	Dallmayr
Cafe Picco	Espresso	7.95	100	Hochland

Mein Kaffeeladen bietet folgende Kaffees an:				
Kaffeename	Kaffeessorte	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.3	75	Dallmayr
Brasil Mild	Bohnenkaffee	6.72	5	Tchibo
Brasilian	Pulverkaffee	5.04	60	Dallmayr
Gran Gusto	Espresso	5.88	145	Cellini
Cafe Range	Schonkaffee	8.77	260	Hochland
Beste Bohne	Bohnenkaffee	5.57	225	Dallmayr
Feine Milde	Schonkaffee	7.61	185	Hochland
Mocca Gold	Bohnenkaffee	7.93	290	Dallmayr
Cafe Picco	Espresso	8.35	230	Hochland

Abbildung 3.28: Vor und nach der Ausführung der Methode *transRestlicheKaffees()*

Bemerkung:

Transaktionen können mit Hilfe der Methode *rollback()* wieder zurückgesetzt werden. Dies wird benötigt, wenn eine Transaktion fehlgeschlagen ist und somit eine *SQLException* ausgeworfen wird. Diese Ausnahme sollte im *catch*-Zweig mit der Methode *rollback()* aufgefangen werden. Weitere Einzelheiten können in *The Java Tutorial Continued*“ auf den Seiten 827ff in einem Beispielprogramm studiert werden.

3.4 Verwendung eines JOINS

Der JOIN ist eine weitere fundamentale Operation, wenn es um die Verknüpfung von mehreren Tabellen geht. Um die Verwendung eines JOINS zu verdeutlichen, benötigst du deshalb eine Datenbank mit mindestens zwei Tabellen.

Übung 3.17:

Erstelle eine Kopie der Datenbank *Kaffeeladen2* unter dem Namen *Kaffeehaus2*.

Nun wirst du in der Datenbank *Kaffeehaus2* nur die Daten über alle Kaffees abfragen, die vom Anbieter „Cellini,“ geliefert werden. Hierzu wird wiederum das *Statement*-Objekt *stmt* verwendet, um mit dessen Methode *executeQuery()* Daten aus der Datenbank zu lesen.

Die Methode *executeQuery()* erwartet einen SQL-String in Form einer gültigen SELECT-Anweisung und gibt ein Objekt *results* vom Typ *ResultSet* zurück, das die Ergebnismenge repräsentiert.

In dieser Abfrage sucht die SQL-Anweisung an die Datenbank *Kaffeehaus1*

```
SELECT Kaffee.Name, Sorte.Sorte, Kaffee.Preis, Anbieter.Name
FROM Kaffee, Sorte, Anbieter
WHERE Anbieter.AnbieterID = Kaffee.AnbieterNr
AND Sorte.SorteID = Kaffee.SorteNr
AND Anbieter.Name LIKE 'Dallmayr'
```

zuerst aus dem Verbund der drei Tabellen *Kaffee*, *Anbieter* und *Sorte* und alle Datensätze, bei denen die *Anbieter.AnbieterID* mit der *Kaffee.AnbieterNr* und auch *Sorte.SorteID* mit der *Kaffee.SorteNr* übereinstimmen. Zum Schluss werden hieraus die Datensätze, deren *Anbieter.Name* gleich der Firma 'Dallmayr' ist, herausgefiltert.



```
BlueJ: Terminal Window - Kaffee02g
Mein Kaffeeladen bietet folgende Kaffees an:
Kaffeename      SorteNr  Preis  Anzahl  AnbieterNr
-----
Espresso dOro   1        7.3    0        1
Brasil Mild     2        6.4   50        3
Brazilian       3        4.8    0        1
Gran Gusto      1        5.6   25        2
Cafe Range      4        8.35  60        4
Beste Bohne     2        5.3   45        1
Colombian       3        7.25   0        3

Cellini liefert folgende Kaffees:
Kaffeename      Kaffeesorte  Preis  Anbieter
-----
Espresso dOro   Espresso     7.3    Dallmayr
Beste Bohne     Bohnenkaffee 5.3    Dallmayr
Brazilian       Pulverkaffee 4.8    Dallmayr
```

Abbildung 3.29: Terminalfenster zeigt alle Kaffees des Anbieters „Dallmayr.“

Bemerkung:

Die Tabelle *Kaffee* der Datenbank *Kaffeehaus2* enthält nun die Fremdschlüssel *SorteNr* und *AnbieterNr*. Dementsprechend musst du auch die Methode *befrageResults()* ändern.

Übung 2.17:

Speichere das Projekt *Kaffee02* unter dem Namen *Kaffee02g*. Implementiere eine Methode *holDallmayr()*, die alle von der Firma „Dallmayr“ gelieferten Kaffees auflistet. Vergleiche dazu Abbildung 3.29