

4 Einfaches Kaffeehaus

In diesem Kapitel wirst du die Kenntnisse aus den vorhergehenden Kapiteln anwenden. Du wirst mit Hilfe von grafischen Oberflächen in einer Datenbank folgende Aufgaben verrichten:

- a) Datensätze eingeben,
- b) Datensätze suchen,
- c) Datensätze ändern,
- d) Datensätze löschen.

Für den Einstieg wirst du in diesem Kapitel eine Datenbank verwenden, die nur aus der Tabelle *Kaffee* besteht. Im Prinzip könntest du die Datenbanken *Kaffeeladen1* bzw. *Kaffeehaus1* verwenden. Da du aber von einer grafischen Oberfläche auf diese Datenbank zugreifst, solltest du in phpMyAdmin eine neue, leere Datenbank *Kaffeehaus3* anlegen, die du im Laufe dieses Kapitels füllen, abfragen, ändern und löschen wirst.

4.1 Die Datenbank Kaffeehaus3

Übung 4.1:

Lege eine neue Datenbank *Kaffeehaus3* an, die die Tabelle *Kaffee* enthält, wie in den Abbildung 4.1 dargestellt. Die Datenfelder der Tabelle *Kaffee* bleiben vorerst leer, da diese mit Hilfe der Java-Anwendung gefüllt werden sollen.

Feld	Typ	Länge/Set*	Kollation	Attribute	Null	Standard**	Extra
KaffeeID	INT				not nul		auto_increment
Name	VARCHAR	100			not nul		
SorteNr	VARCHAR	100			not nul		
Preis	DECIMAL	3, 2			not nul		
Anzahl	INT				not nul	0	
Anbieter	VARCHAR	100			not nul		

Tabellen-Kommentar:

Tabellentyp:

Kollation:

Abbildung 4.1: Anlegen der Datenbank *Kaffeehaus3*

4.2 Die grafische Benutzeroberfläche

Nun wirst du dich der grafischen Benutzeroberfläche widmen. Die Abbildung 4.2 zeigt eine mit Java-Swing gestaltete, einfache GUI.

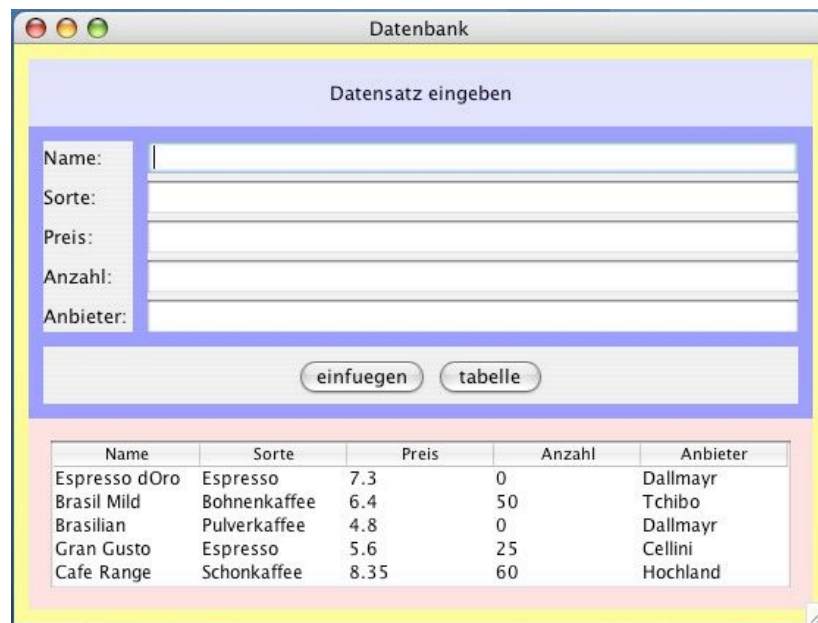


Abbildung 4.2: Grafische Benutzeroberfläche eines Eingabeformulars

Die Gestaltungsmöglichkeiten mit Swing werden in *The JFC Swing Tutorial, Second Edition* von Kathy Walrath, u. a. oder in ähnlichen Büchern beschrieben.

In Swing werden die Fensterkomponenten in eine Art Container, so genannte Panels, gepackt. In der Abbildung 4.2 bezeichne ich den Teil des gesamten Fensters, in dem gezeichnet werden kann (also ohne der Menüleiste) als Hauptpanel *mainP*. Dieser Teil ist der gelbe Bereich.

Dieses Hauptpanel ist wiederum unterteilt in das

- hellblaue Namenpanel *namenP* mit einem Label,
- dunkelblaue Abfragepanel *abfrageP* mit den vier Labels, den vier zugehörigen Textfeldern und den beiden Buttons,
- hellrote Tabellenpanel *tabellenP*, in das die eingefügten Daten in Tabellenform eingetragen werden.

Die beiden Buttons erhalten je einen *ActionListener*, der das Anklicken überwacht und dann die entsprechenden Aktionen auslöst.

Übung 4.2:

Hole vom Schulserver das Projekt *EingabeGUI* und speichere dieses als Projekt *Eingabe01*. Studiere den Quelltext der Klasse *Ansicht*.

- Suche die Komponenten *namenP*, *einfuegenP* und *tabellenP*.
- Suche die Komponenten *nameB*, *preisTF* und *einfuegenB*.

In vielen grafischen Oberflächen wird die Anordnung der Komponenten durch Angaben absoluter Koordinaten vorgenommen. Da jedoch Java-Programme auf unterschiedlichen Plattformen laufen sollen, ist eine solche Vorgehensweise nicht sinnvoll. Zur Anordnung der Komponenten verwendet man in Java verschiedene so genannte Layoutmanager. Diese sorgen dafür, dass die Benutzeroberfläche auf allen Plattformen das gleiche Aussehen hat (weitestgehend).

Ich verwende in diesem Beispiel den *GridLayout-Manager*, den *BoxLayout-Manager* und den *BorderLayout-Manager*. Im *GridLayout* werden die Komponenten innerhalb eines rechteckigen Gitters angeordnet. Die Parameter beim Aufruf des Konstruktors bestimmen die vertikale und die horizontale Anzahl der Elemente. Im *BoxLayout* werden die Komponenten standardmäßig untereinander in der benötigten Größe angeordnet. Weitere Einzelheiten werden in den oben genannten Büchern beschrieben.

4.2.1 Das Abfragepanel

Die folgenden Übungen werden dich schrittweise vertraut machen mit der Programmierung von grafischen Oberflächen in Java. Zum Schluss wirst du eine Oberfläche programmiert haben wie in Abbildung 4.2 dargestellt. Verwende so oft wie möglich das Verfahren „*Copy and Paste*“!

Übung 4.3:

- a) Ändere in der Methode *setTitle()* das Argument „Datenbank“ um in „MySQL-Datenbank“.
- b) Ändere im Namenpanel *namenP* den Text „Kaffee-Datenbank“ um in „Datensatz eingeben“.
- c) Ändere in der Methode *setBounds()* die ursprünglichen Werte (50, 50, 700, 400) in (300, 200, 700, 400) bzw. in (50, 50, 600, 200). Formuliere schriftlich, welche Veränderungen dadurch stattfinden. Setze in der Methode *setBounds()* wieder die ursprünglichen Werte ein.

Übung 4.4:

- a) Füge im Einfügenpanel *ein fuegenP* zusätzlich die Labels und Textfelder für den Preis, die Anzahl und den Anbieter hinzu. Vergiss nicht den zusätzlichen Eintrag im Variablenkatalog.
- b) Füge im Buttonpanel *buttonP* den zweiten Button *tabelle* hinzu. Vergiss

nicht den zugehörigen *ActionListener*

Nach Abschluss dieser Übungen sollte das Fenster des Projekts *Eingabe01* der Abbildung 3.3 gleichen. Allerdings fehlt in dieser GUI noch die Tabelle, in die die Ergebnisse aus den Datenbankzugriffen später eingetragen werden.



Abbildung 4.3: Grafische Benutzeroberfläche von Projekt *Eingabe01*

4.2.2 Die Tabelle

Es ist sinnvoll, die Ergebnisse eines Datenbankzugriffs in Tabellenform innerhalb der GUI darzustellen. Java-Swing bietet hierzu die Klasse *JTable*, die eine mehrzeilige und mehrspaltige Darstellung von Daten ermöglicht.

Übung 4.5:

Hole vom Schulserver das Projekt *TabelleGUI* und speichere dieses als *Tabelle01*. Die Abbildung 4.4 zeigt die entsprechende Tabelle.

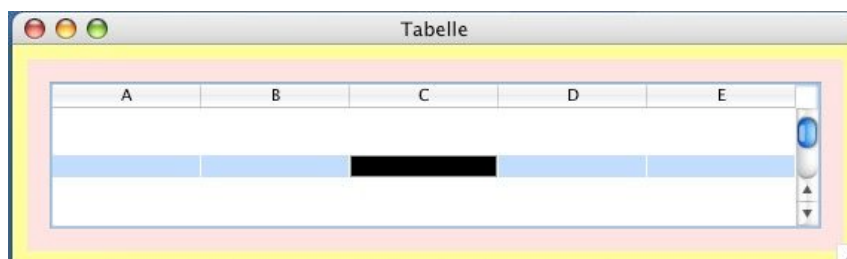


Abbildung 4.4: Das Projekt *Tabelle01* stellt eine einfachen Tabelle dar

Übung 4.6:

- a) Klicke z. B. in der Spalte C in die dritte Zelle. Nun wird die gesamte Zeile selektiert und die entsprechende Zelle als editierbar markiert. Ein Doppelklick auf diese Zelle ermöglicht das Schreiben in diese Zelle. Schreibe irgendetwas in einige Zellen.
- b) Klicke in die Zelle mit dem Buchstaben A und bewege den Cursor mit gedrückter Maustaste nach rechts. Du kannst nun die Spalten einer Tabelle neu sortieren.
- c) Bewege den Cursor zwischen die zwei Spalten A und B. Er verwandelt sich in einen Doppelpfeil und die Spaltenbreite lässt sich nun verändern.

Die Tabelle wird erzeugt in der Methode *tabellenPErstellen()*. Im Projekt *Eingabe01* hast du bereits ein *ScrollPane* erstellt, in das nun die Tabelle gelegt werden kann.

```
private Container tabellenPErstellen()
{
    tabellenP = new JPanel();
    tabellenP.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

    //diese drei Anweisungen ergeben die gesamte Tabelle
    tabelle = new JTable(10, 5);
    tabelle.setPreferredSize(new Dimension(500, 80));
    scrollPane = new JScrollPane(tabelle);

    tabellenP.add(scrollPane);
    if (farbig) tabellenP.setBackground(new Color(255, 220, 220));
    return tabellenP;
}
```

Abbildung 4.5: Die Methode *tabellenPErstellen()*

In der Methode *tabellenPErstellen()* wird eine Tabelle mit zehn Zeilen und fünf Spalten erzeugt. Die Kopfzeilen der Tabelle sind bereits beschriftet mit den Buchstaben A, B, C, D und E. Eine bestimmte voreingestellte Größe erhält die Tabelle mit der Methode *setPreferredSize()*. Im Allgemeinen ist die Größe einer Tabelle allerdings nicht vorhersagbar, da Datensätze eingefügt bzw. gelöscht werden können. Deswegen wird in der Regel eine Tabelle auf einem *JScrollPane* dargestellt.

Übung 4.7:

Erzeuge eine Tabelle mit vier Spalten. Beobachte die Scrollbars.

Zur Erzeugung dieser Tabelle wurde der einfachste Konstruktor verwendet, der die Anzahl von Zeilen und Spalten als Parameter enthält. Es ist auch möglich, die Tabelle mit bereits vorhandenen Daten zu füllen. Wir benötigen nun die Spaltennamen und die Daten, die in die einzelnen Zellen geschrieben werden.

```
String[] spaltenNamen = {"Name",
                        "Sorte",
                        "Preis",
                        "Anzahl",
                        "Anbieter"};

Object[][] werte = {
    {"Espresso dOro", "Espresso", new Double(7.30),
                                     new Integer(0), "Dallmayr"},
    {"Brasil Mild", "Bohnenkaffee", new Double(6.40),
                                     new Integer(50), "Tchibo"},
    {"Brasilian", "Pulverkaffee", new Double(4.80),
                                     new Integer(0), "Dallmayr"},
    {"Gran Gusto", "Espresso", new Double(5.60),
                                     new Integer(25), "Cellini"},
    {"Cafe Range", "Schonkaffee", new Double(8.35),
                                     new Integer(60), "Hochland"}
};
```

..... weitere Anweisungen

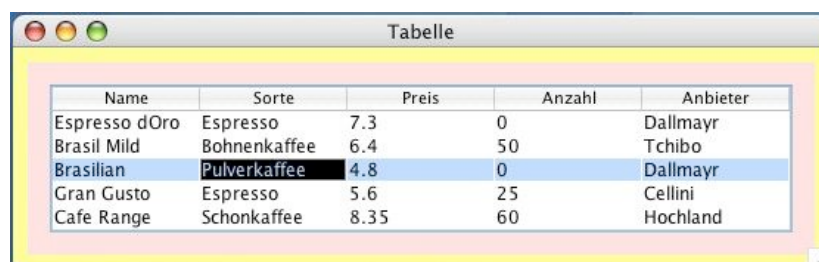
```
tabelle = new JTable(werte, spaltenNamen);
```

..... weitere Anweisungen

Abbildung 4.6: Zusätzlicher Quelltext in der Methode *tabellenPERstellen()*

Übung 4.8:

Füge den in Abbildung 4.6 dargestellten Quelltext in die Methode *tabellenPERstellen()* an den entsprechenden Stellen hinzu.



Name	Sorte	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

Abbildung 4.7: Die Tabelle wurde mit Spaltennamen und Daten gefüllt

Übung 4.9:

Speichere dein Projekt *Eingabe01* unter dem Namen *Eingabe02*. Verändere die Methode *tabellenPERstellen()* analog zu *Tabelle01*. Verwende *Copy and Paste!* Nun solltest du eine ähnliche grafische Benutzeroberfläche erstellt haben wie in der Abbildungen 4.2 dargestellt.

Beachte, dass diese Daten in der *JTable* *tabelle* nicht aus der Datenbank *Kaffeehaus2* stammen, sondern dass du diese sozusagen „per Hand“ in die Tabelle hineingeschrieben hast. Wie die Daten aus der Datenbank gelesen werden, lernst du in den folgenden Kapiteln.

4.3 Schnittstelle zwischen Ansicht und DBAbfrage

Unser Projekt, das einen einfachen Kaffeeladen modelliert, wird in zwei größere Teile zerlegt: Der eine Teil ermöglicht dem Benutzer, durch Drücken von Buttons und Eingabe in die Textfelder nach bestimmten Kaffees in der Datenbank zu suchen. Der andere Teil ermöglicht mit entsprechenden SQL-Anweisungen die gewünschten Datensätze in der Datenbank zu bearbeiten.

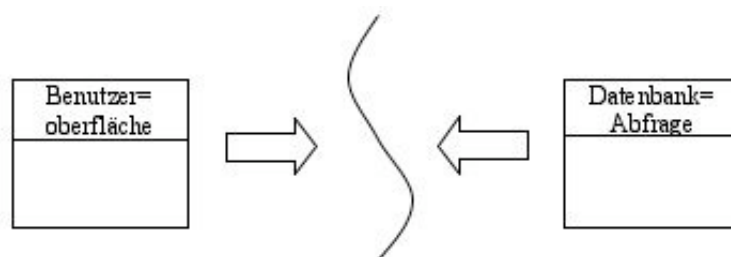


Abbildung 4.8: Festlegung der Schnittstelle

Die Abbildung 4.8 zeigt, dass für beide Teile klar definiert sein muss, wie sie den jeweils anderen Teil benutzen können – die Schnittstelle zwischen ihnen muss definiert werden. In größeren Projekten muss es also klare Richtlinien geben, welches Entwicklungsteam für welche Aufgaben zuständig ist und wie die verschiedenen Teile schließlich in der Gesamtanwendung zusammenspielen sollen. Es wird also notwendig sein, die Schnittstellen festzulegen, bevor an der Implementierung der Teile gearbeitet werden kann.

Du wirst in den folgenden Kapiteln die Benutzeroberfläche mit den Datenbankzugriffen verbinden. Dazu muss die Schnittstelle zwischen diesen beiden Klassen definiert werden. Du musst also festlegen, welche Funktionen

die Buttons besitzen und wie aus den Textfeldern die entsprechenden Daten herausgelesen und in die Tabelle der GUI eingetragen werden.

Das Eingabeformular soll also folgende Aufgaben übernehmen:

- a) Zu Beginn werden alle Kaffees aus der Datenbank *Kaffeehaus3* in der Tabelle aufgelistet.
Methode: *holeDaten()*
- b) Wird in die Textfelder ein neuer Kaffee eingetragen, soll dieser Datensatz mit Hilfe des Buttons *ein fuegen* in die Datenbank *Kaffeehaus3* eingefügt werden.
Methode: *neuDaten()*
- c) Zur Darstellung aller Datensätze benötigt die Tabelle die Spaltennamen und die Daten.
Methoden: *gibSpaltennamen()*, *gibWerte()*
- d) Mit Hilfe des Buttons *tabelle* wird die Tabelle aktualisiert und listet alle Datensätze auf.
- e) Aus didaktischen Gründen habe ich mich entschieden, für die Aktualisierung der Tabelle vorerst einen zweiten Button *tabelle* zu verwenden. Aus benutzerfreundlichen Gründen ist es jedoch sinnvoll, dessen Funktion dem Button *ein fuegen* zuweisen.

Mit Hilfe dieses Aufgabenkatalogs der Klasse *Ansicht* kannst du nun folgendes Modell der Klasse *DBAbfrage* erstellen.



Abbildung 4.9: Modell der Klasse *DBAbfrage*

Der Methode *holeDaten()* wird eine SELECT-Anweisung als Parameter übergeben, die alle Daten aus der Tabelle *Kaffee* herausliest. Der Methode *neuDaten()* wird eine INSERT-Anweisung als Parameter übergeben, die einen neuen Datensatz in die Tabelle *Kaffee* einfügt. Die beiden Methoden

gibSpaltennamen() und *gibWerte()* liefern diejenigen Spaltennamen und Attributwerte, die in der Tabelle der GUI aufgelistet werden sollen. Alle weiteren Methoden in der Klasse *DBAbfrage* werden als *private* definiert, sind somit für die Klasse *Ansicht* nicht sichtbar und können deshalb auch von der Klasse *Ansicht* nicht verwendet werden.

4.4 Implementierung der Klasse *DBAbfrage*

In diesem Abschnitt wirst du die Klasse *DBAbfrage()* implementieren. Die wichtigsten Teile hast du bereits im Kapitel 3 *Datenbankzugriff in Java* kennen gelernt.

Übung 4.10:

Speichere das Projekt *Kaffee02c* (aus Kapitel 3 *Datenbankzugriff in Java*) unter dem Namen *Eingabe03*. Ändere gemäß den folgenden Abschnitten in Kapitel 4.4 die Klasse *DBAbfrage*.

Zuerst müssen die benötigten Datenfelder deklariert werden. Da du in die Datenbank *Kaffeehaus3* noch keine Datensätze eingefügt hast, wird im Konstruktor vorerst noch die Datenbank *Kaffeehaus1* übergeben werden.

```
private DBVerbindung dbVerbindung;  
  
private String[] spaltennamen;  
private Object[][] werte;  
  
public DBAbfrage()  
{  
    dbVerbindung = new DBVerbindung("Kaffeehaus1", "ralph", "ralph");  
}
```

Abbildung 4.10: Datenfelder und Konstruktor der Klasse *DBAbfrage*

4.4.1 Die Methode *holeDaten()*

Diese Methode muss nur geringfügig geändert werden.

```
public void holeDaten(String fragen)  
{  
    //Beispiel fuer die Parametereingabe:  
    //String fragen = "SELECT Name, Sorte, Preis, Anzahl, Anbieter " +  
    //                "FROM Kaffee";
```

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

try {
    ResultSet results = stmt.executeQuery(fragen);
    befrageResults(results);
}
catch (SQLException sqle) {
    System.out.println(sqle.toString());
}

dbVerbindung.schliesseVerbindung();
}
```

Abbildung 4.11: Die Methode *holeKaffee()* in der Klasse *DBAbfrage*

Wie bereits bei der Festlegung der Schnittstelle besprochen, wird dieser Methode eine SELECT-Anweisung als Parameter übergeben. Damit wird ein *results* erstellt, aus dem mit Hilfe der Methode *befrageResults()* alle benötigten Daten für die Tabelle der GUI herausgelesen werden.

4.4.2 Die Methode *befrageResults()*

Diese Methode muss komplett umgeschrieben werden. Im Projekt *Eingabe02* hast du die Tabelle mit Daten gefüllt, die per Hand eingegeben worden sind. Nun sollen diese Daten aus der Datenbank geholt und in der GUI dargestellt werden.

Folgende Probleme müssen hierbei berücksichtigt werden

1. Mit Hilfe der SQL-Abfrage
`ResultSet results = stmt.executeQuery(fragen);`
erhält man den *ResultSet results*, der alle Daten aus der Tabelle *Kaffee* enthält.
2. Die Daten werden mit *results.getString()*, etc. aus *results* gelesen.
3. Die Tabelle benötigt das zweidimensionale Array *werte = new Object[anzZeilen][anzSpalten]*, das die entsprechenden Werte enthält. Dazu wird der *results* mit seiner Methode *next()* durchlaufen.
4. Die Tabelle benötigt das Array *spaltennamen = new String[anzSpalten]*, das die entsprechenden Spaltennamen enthält. Diese befinden sich in den Metadaten von *results*. Die Metadaten erhält man mit
`ResultSetMetaData resultsMD = results.getMetaData();`

Die Spaltennamen erhält man anschließend mit `resultsMD.getColumnLabel()`.

Die ersten drei Punkte wurden bereits im Kapitel 3 im Projekt *Kaffee02* besprochen. Für den vierten Punkt musst du den kompletten *try*-Zweig ersetzen. Abbildung 4.12 zeigt bereits den gesamten Quelltext des *try*-Zweigs. In den nachfolgenden Übungen kannst du dir die Zusammenhänge erarbeiten.

```
try {
    //Sucht nach Metadaten in results
    ResultSetMetaData resultsMD = results.getMetaData();
    //Anzahl der Spalten
    int anzSpalten = resultsMD.getColumnCount();
    //Spaltennamen
    spaltennamen = new String[anzSpalten];
    for (int i = 0; i < anzSpalten; i++) {
        spaltennamen[i] = resultsMD.getColumnLabel(i+1);
    }
    //Anzahl der Zeilen (Groesse des ResultSets)
    results.last(); //Cursor nun in der letzten Zeile von results
    int anzZeilen = results.getRow();
    results.beforeFirst(); //Cursor wieder vor der ersten Zeile von results
    //Daten aus den Zellen von results
    werte = new Object[anzZeilen][anzSpalten];
    int i = 0;
    while (results.next()) {
        for (int j = 0; j < anzSpalten; j++) {
            werte[i][j] = results.getString(j+1);
        }
        i++;
    }
}
```

Abbildung 4.12: Gesamter Quelltext im *try*-Zweig der Methode *befrageResults()*

Die folgenden Übungen zeigen dir, wie du aus dem *ResultSet results* alle Werte ermitteln kannst, die für die Tabelle der GUI benötigt werden.

Wie bereits erwähnt, benötigt die Tabelle der GUI die Anzahl der Spalten und die zugehörigen Spaltennamen. Diese befinden sich in den Metadaten von *results*, die du mit `ResultSetMetaData resultsMD = results.getMetaData();` erhältst. Die Spaltennamen liefert die Methode `getColumnLabel()`. Beachte auch, dass das erste Element von *resultMD* den Index 1 hat, während das *Array spaltennamen* mit dem Index 0 beginnt.

```
//Sucht nach Metadaten in results
```

```
ResultSetMetaData resultsMD = results.getMetaData();
//Anzahl der Spalten
int anzSpalten = resultsMD.getColumnCount();
//Spaltennamen
spaltennamen = new String[anzSpalten];
for (int i = 0; i < anzSpalten; i++) {
    spaltennamen[i] = resultsMD.getColumnLabel(i+1);
    System.out.print(spaltennamen[i] + "\t");
}
System.out.println();
System.out.println();
```

Abbildung 4.13: Die Methode *befrageResults()* liefert hiermit die Spaltennamen

Übung 4.11:

Hole dir vom Schulserver das Projekt *Result*. Da der *try*-Zweig keine Anweisungen enthält, ist dieses Projekt nicht kompilierbar. Implementiere den Quelltext aus Abbildung 4.13 im *try*-Zweig der Methode *befrageResults()*. Der Aufruf der Methode *holeDaten()* liefert nun alle Spaltennamen. (Projekt *Result01*)



Abbildung 4.14: Die Methode *holeDaten()* liefert die Spaltennamen

In den nächsten Anweisungen musst du die Anzahl der Zeilen ermitteln. Die Methode *getRow()* liefert diejenige Zeilennummer, in der der interne Cursor des *results* aktuell steht, wobei erste Zeile die Nummer 1 hat, die zweite die Nummer 2 usw. Um die Anzahl der Zeilen zu bestimmen, wird also der Cursor mit Hilfe der Methode *last()* in die letzte Zeile gesetzt, deren Zeilennummer ermittelt und anschließend wieder vor die erste Zeile gesetzt. Somit kann später die Methode *next()* den *result* wieder korrekt durchlaufen.

```
//Anzahl der Zeilen (Groesse des ResultSets)
results.last(); //Cursor nun in der letzten Zeile von results
int anzZeilen = results.getRow();
results.beforeFirst(); //Cursor wieder vor der ersten Zeile von results#
System.out.println("Anzahl der Zeilen: " + anzZeilen);
System.out.println("Anzahl der Spalten: " + anzSpalten);
```

```
System.out.println();  
System.out.println();
```

Abbildung 4.15: Die Methode *befrageResults()* liefert nun Anzahl der Zeilen bzw. Spalten

Übung 4.12:

Implementiere zusätzlich den Quelltext aus Abbildung 3.15 im *try*-Zweig der Methode *befrageResults()*. Der Aufruf der Methode *holekaffees()* liefert nun auch die Anzahl der Zeilen und Spalten.

(Projekt *Result02*)



Abbildung 4.16: Die Methode *holeDaten()* liefert Anzahl der Zeilen und Spalten

Die letzten Anweisungen der Methode *befrageResults()* holen aus dem *ResultSet results* die Werte und schreiben diese in ein zweidimensionales Array *Object[anzZeilen][anzSpalten] daten*.

```
//Werte aus den Zellen von results  
werte = new Object[anzZeilen][anzSpalten];  
int i = 0;  
while (results.next()) {  
    for (int j = 0; j < anzSpalten; j++) {  
        werte[i][j] = results.getString(j+1);  
        System.out.print(werte[i][j] + "\t");  
    }  
    System.out.println();  
    i++;  
}
```

Abbildung 4.17: Die Methode *befrageResults()* liefert nun alle Werte

Übung 4.13:

Implementiere zusätzlich den Quelltext aus Abbildung 4.17 im *try*-Zweig der Methode *befrageResults()*. Der Aufruf der Methode *holeDaten()* liefert nun alle Daten, die die Tabelle der GUI benötigt.

(Projekt *Result03*)

Name	Sorte	Preis	Anzahl	Anbieter
Anzahl der Zeilen: 9				
Anzahl der Spalten: 5				
Espresso dOro	Espresso	7.30	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 4.18: Die Methode *befrageResults()* liefert alle Werte

Nun solltest du den Quelltext aus Abbildung 4.12 verstanden haben und diesen in deinem Projekt *Eingabe03* in den *try*-Zweig der Methode *befrageResults()* implementieren.

4.4.3 Die Methoden *gibSpaltennamen()* und *gibDaten()*

Die beiden Methoden *gibSpaltennamen()* und *gibDaten()* liefern nun die von der Tabelle benötigten Arrays.

```
public String[] gibSpaltennamen()
{
    return spaltennamen;
}

public Object[][] gibWerte()
{
    return werte;
}
```

Abbildung 4.19: Die Methoden *gibSpaltennamen()* und *gibWerte()* in der Klasse *DBAbfrage*

Bemerkung:

Falls noch vorhanden lösche die nun beiden überflüssigen Methoden *neuColombian()* und *neuMoccaRange()* aus dem ehemaligen Projekt *Kaffee02c*

Vergleiche nun an Hand der folgenden Abbildungen dein bisheriges Projekt *Eingabe03*.

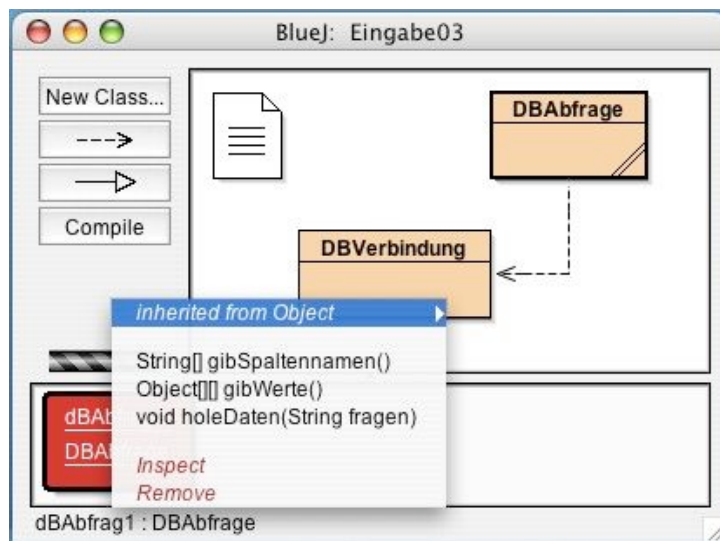


Abbildung 4.20: Klassendiagramm von *Eingabe03*



Abbildung 4.21: Aufruf der Methode *holeDaten()* mit dem Parameter "SELECT Name, Sorte, Preis, Anzahl, Anbieter FROM Kaffee"

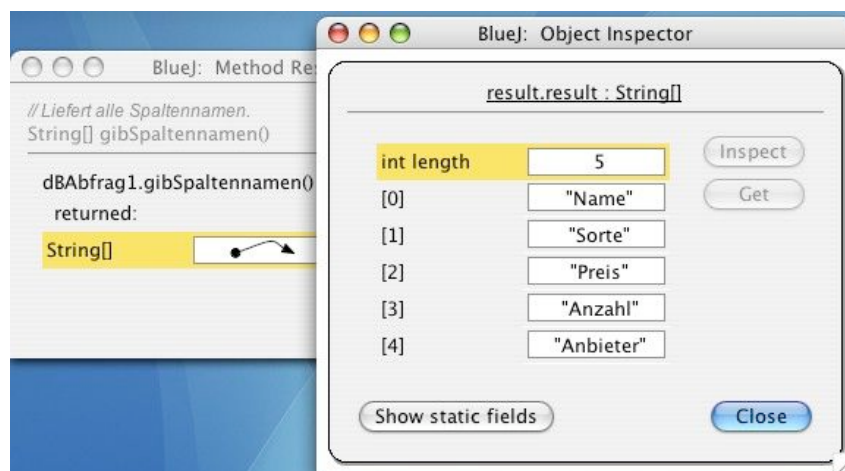


Abbildung 4.22: Die Methode *gibSpaltennamen()* liefert alle Spaltennamen

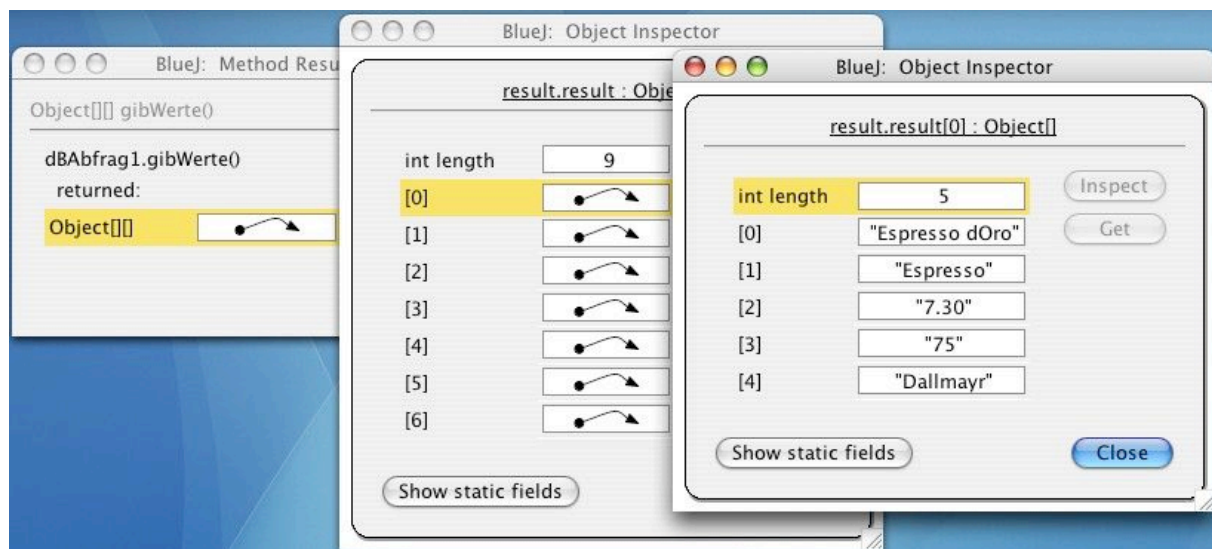


Abbildung 4.23: Die Methode *gibWerte()* liefert hier den ersten Datensatz

Du hast nun die Klasse *DBAbfrage* so implementiert, dass die beiden Methoden *gibSpaltennamen()* und *gibWerte()* aus der Datenbank *Kaffeehaus1* genau die Daten für die *JTable* *tabelle* liefern, die du im Projekt *Eingabe02* noch „per Hand“ in *tabelle* eingetippt hast.

4.4.4 Die Methode *neuDaten()*

Bemerkung:

Mit Hilfe dieser Methode *neuDaten()* kannst du nun Datensätze in eine Datenbank eingeben. Deshalb solltest du im Konstruktor der Klasse *DBAbfrage* ab jetzt die Datenbank *Kaffeehaus3* angeben.

Wie bereits bei der Festlegung der Schnittstelle besprochen, wird dieser Methode eine INSERT-Anweisung als Parameter übergeben.

```
public void neuDaten(String updKaffee)
{
//Beispiel fuer die Parametereingabe:
//String updKaffee = "INSERT INTO Kaffee(Name, Sorte, Preis, Anbieter) " +
//                "VALUES ('Espresso dOro', 'Espresso', 7.30, 'Dallmayr')";

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

    try {
        stmt.executeUpdate(updKaffee);
    }
}
```

```

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

dbVerbindung.schliesseVerbindung();
}

```

Abbildung 4.24: Die Methode *neuKaffee()* in der Klasse *DBAbfrage*

Bemerkung:

Die Datenfelder *KaffeeID* und *Anzahl* der Tabelle *Kaffee* müssen in der INSERT-Anweisung nicht aufgeführt werden, *KaffeeID* ist *auto_increment* und *Anzahl* wird mit dem Standardwert 0 belegt wird. In der Klasse *Ansicht* wirst du jedoch in der INSERT-Anweisung das Datenfeld *Anzahl* verwenden.

Anschließend kannst du in die Datenbank *Kaffeehaus3* die beiden Datensätze aus Abbildung 4.26 eingeben.



Abbildung 4.25: Aufruf der Methode *neuDaten()* mit dem Parameter "INSERT INTO Kaffee(Name, Sorte, Preis, Anbieter) VALUES ('Espresso dOro', 'Espresso', 7.30, 'Dallmayr')"

KaffeeID	Name	Sorte	Preis	Anzahl	Anbieter
1	Espresso dOro	Espresso	7.30	0	Dallmayr
2	Brasil Mild	Bohnenkaffee	6.40	0	Tchibo

Abbildung 4.26: Tabelle *Kaffee* enthält bereits zwei Datensätze

4.5 Datensätze eingeben

In diesem Kapitel wirst du die Schnittstelle zwischen der Klasse *Ansicht* und der Klasse *DBAbfrage* vervollständigen. Anschließend bist du in der Lage, mit Hilfe des Formulars *Datensätze eingeben* die Datenbank zu füllen

Übung 4.14:

Speichere dein Objekt *Eingabe03* unter dem Namen *Eingabe04*. Füge mit Hilfe von *Edit > Add Class from File...* die Klasse *Ansicht* aus deinem Projekt *Eingabe02* hinzu. Ändere gemäß den folgenden Abschnitten in Kapitel 4.5 die Klasse *Ansicht*.

4.5.1 Die Methode *tabellenPERstellen()*

Die Klasse *DBAbfrage* stellt nun alle Methoden zur Verfügung, um in der Tabelle der GUI die Daten aus der Datenbank aufzulisten. Zuerst erstellst du eine SELECT-Anweisung, die der Methode *holeDaten()* des Objekts *dbAbfrage* als Parameter mitgeben wird. Dieses Objekt liefert anschließend die *spaltennamen* und *werte*. Daraus wird dann nach bekanntem Muster die Tabelle erzeugt und mit den Daten gefüllt.

```
private Container tabellenPERstellen()
{
    String qryKaffee = "SELECT Name, Sorte, Preis, Anzahl, Anbieter " +
        "FROM Kaffee";

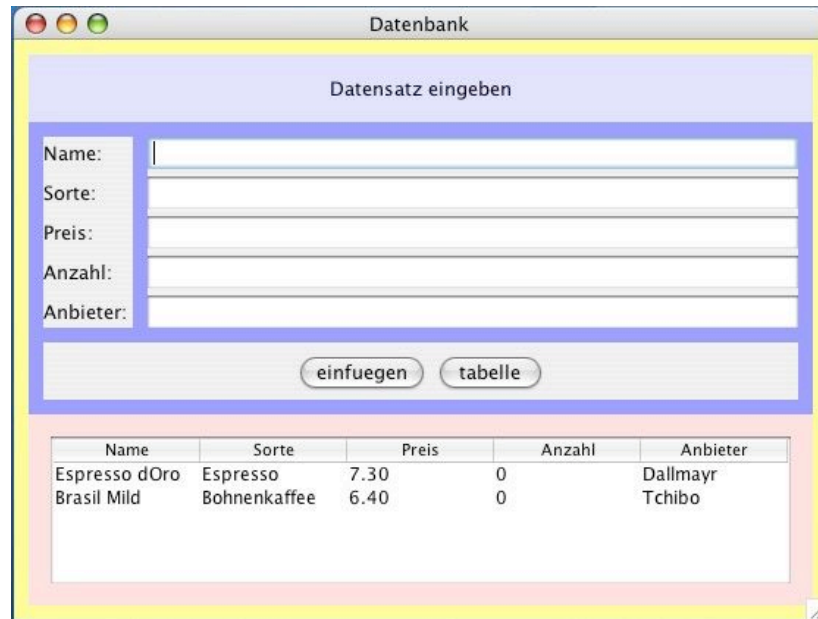
    dbAbfrage = new DBAbfrage();
    dbAbfrage.holeDaten(qryKaffee);
    String[] spaltenNamen = dbAbfrage.gibSpaltennamen();
    Object[][] werte = dbAbfrage.gibWerte();

    tabellenP = new JPanel();

    ..... weitere Anweisungen .....
}
```

Abbildung 4.27: Die Methode *tabellenPERstellen()* in der Klasse *Ansicht*

Nun kannst du bereits dein Objekt *Eingabe04* ausprobieren. Beim Öffnen des Eingabeformulars werden in der Tabelle die beiden bereits eingegebenen Datensätze aus der Datenbank *Kaffeehaus3* dargestellt.



Name	Sorte	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.30	0	Dallmayr
Brasil Mild	Bohnenkaffee	6.40	0	Tchibo

Abbildung 4.28: Das Formular *Datensatz eingeben* zeigt die beiden Datensätze aus der Datenbank *Kaffeehaus2*

4.5.2 Die Methode *einfuegenBActionPerformed()*

Die Methode *einfuegenBActionPerformed()* holt aus den Textfeldern die Einträge und erstellt daraus die SQL-Anweisung *updKaffee*. Dieser String wird der Methode *neuDaten()* des Objekts *dbAbfrage* als Parameter mitgegeben.

```
/** Reagiert auf Druecken von einfuegenB. */
private void einfuegenBActionPerformed(ActionEvent e)
{
    String name = nameEinTF.getText();
    String sorte = sorteEinTF.getText();
    double preis = Double.parseDouble(preisEinTF.getText());
    int anzahl = Integer.parseInt(anzahlEinTF.getText());
    String anbieter = anbieterEinTF.getText();

    String updKaffee = "INSERT INTO Kaffee(Name, Sorte, Preis, Anzahl,
                        Anbieter) " +
        "VALUES ('" + name + "', '" + sorte + "', " + preis +
        ", " + anzahl + ", '" + anbieter + "')";

    dbAbfrage = new DBAbfrage();
    dbAbfrage.neuDaten(updKaffee);
}
```

Abbildung 4.29: Die Methode *einfuegenBActionPerformed()* in der Klasse *Ansicht*

Abbildung 4.30: Benutzer erstellt zur Zeit den dritten Datensatz

Nach dem Drücken auf den Button *einfuegen*, befindet sich der dritte Datensatz in der Datenbank *Kaffeehaus2*.

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

Abbildung 4.31: Der neue Datensatz wird in der Tabelle *Kaffee* angezeigt

4.5.3 Die Methode *tabelleBActionPerformed()*

Im Prinzip funktioniert die Methode *tabelleBActionPerformed()* genauso wie das Erstellen des Tabellenpanels *tabelleP*. Allerdings wird das ursprüngliche *scrollPane* aus *tabellenP* entfernt und ein neues *scrollpane* zusammen mit der aktualisierten *tabelle* wieder dem *tabellenP* hinzugefügt.

```
private void tabelleBActionPerformed(ActionEvent e)
{
    String qryKaffee = "SELECT Name, AnbieterNr, Sorte, Preis, Anzahl " +
        "FROM Kaffee";

    dbAbfrage = new DBAbfrage();
    dbAbfrage.holeKaffees(qryKaffee);
    String[] spaltenNamen = dbAbfrage.gibSpaltennamen();
    Object[][] daten = dbAbfrage.gibDaten();
}
```

```

tabellenP.remove(scrollPane);
tabelle = new JTable(daten, spaltenNamen);
tabelle.setPreferredScrollableViewportSize(new Dimension(500, 80));
scrollPane = new JScrollPane(tabelle);
tabellenP.add(scrollPane);
pack();
}

```

Abbildung 4.32: Die Methode *tabelleBActionPerformed()* in der Klasse *Ansicht*

4.5.4 Benutzerfreundlichkeit

Wie bereits erwähnt, sollten nun aus benutzerfreundlichen Gründen die beiden Buttons *einfüegen* und *tabelle* zusammengelegt werden.



Abbildung 4.33: Der gerade eingefügte vierte Datensatz wird in der aktualisierten Tabelle gezeigt

Dazu werden die entsprechenden Anweisungen aus der Methode *tabelleBActionPerformed()* in die Methode *einfüegenBActionPerformed()* kopiert. Anschließend kann der Button *tabelle* mit seinen *ActionListener* gelöscht werden. Nun wird die Tabelle automatisch aktualisiert, der gerade in die Datenbank eingefügte Datensatz wird in der Tabelle aufgelistet (vgl. Projekt *Eingabe04a*).

Die beiden Methoden *tabellenPErstellen()* und *einfüegenBActionPerformed()* enthalten noch einige Anweisungen, die in beiden gemeinsam verwendet

werden. Diese können in eine separate Methode *holeMySQLDaten()* zusammengefasst werden(vgl. Projekt *Eingabe04b*).

```
/** Holt sich die aktuellen Daten aus der MySQL-Datenbank. */
private void holeMySQLDaten(String sqlAbfrage)
{
    dbAbfrage = new DBAbfrage();
    dbAbfrage.holeDaten(sqlAbfrage);
    String[] spaltenNamen = dbAbfrage.gibSpaltennamen();
    Object[][] werte = dbAbfrage.gibWerte();
    tabelle = new JTable(werte, spaltenNamen);
    tabelle.setPreferredScrollableViewportSize(new Dimension(500, 80));
    scrollPane = new JScrollPane(tabelle);
}
```

Abbildung 4.34: Die Methode *holeMySQLDaten()* im Projekt *Eingabe04a*

Übung 4.15:

In Kapitel 3.2.1 *Einfügen von Datensätzen* zeigt in Abbildung 3.19 das Terminalfenster eine andere Art der Darstellung der Daten. Fülle die Datenbank *Kaffeehaus2* mit diesen dargestellten Datensätzen.

4.6 Zugriff auf die Datenbank mit Hilfe der Klassen *DBVerbindung* und *DBAbfrage*

In den weiteren Aufgaben

- a) Datensätze suchen,
- b) Datensätze ändern,
- c) Datensätze löschen.

wird nur noch die Benutzeroberfläche angepasst. Die Klassen *DBVerbindung* und *DBAbfrage* bleiben unverändert. Deswegen solltest du jetzt einen Blick auf das Zusammenspiel dieser beiden Klassen werfen.

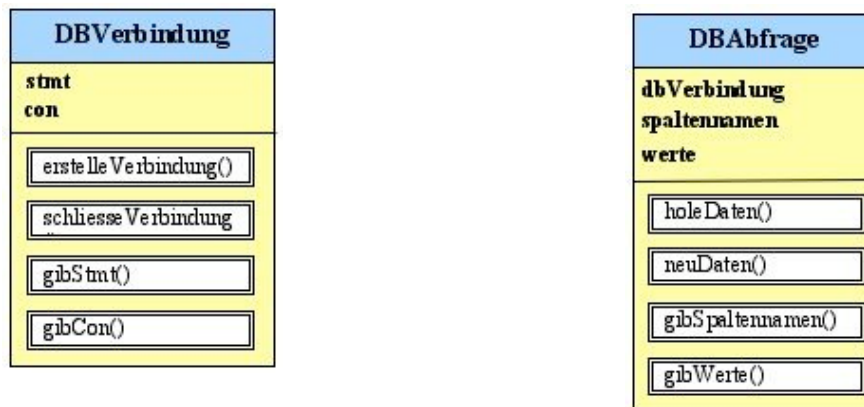


Abbildung 4.35: Modelle der Klassen *DBVerbindung* und *DBAbfrage*

Die Klasse *DBVerbindung* besitzt u.a. die beiden Datenfelder *stmt* und *con*. Diese Anweisungsobjekte werden zur Abfrage und zur Änderung der Datenbank benötigt. Die beiden Methoden *gibStmt()* und *gibCon()* liefern diese Anweisungsobjekte.

Bemerkung:

Das Anweisungsobjekt *con* wird, wie in Kapitel 3 beschrieben, erst bei der Verwendung von Transaktionen benötigt. Deswegen enthält im Projekt *Eingabe04* die Klasse *DBVerbindung* noch nicht das Anweisungsobjekt *con*. Obwohl erst im Kapitel 4.9 *Ändern von Datensätze* Transaktionen besprochen werden, implementiere ich dieses Anweisungsobjekt *con* und die Methode *gibCon()* in der Klasse *DBVerbindung* bereits jetzt.

Das Aufbauen und das anschließende Schließen der Verbindung zur gewählten MySQL-Datenbank werden mit Hilfe der Methoden *erstelleVerbindung()* und *schliesseVerbindung()* durchgeführt.

Die Klasse *DBAbfrage* erzeugt ein Objekt der Klasse *DBVerbindung*. Mit Hilfe dieses Objekts können nun

1. mit der Methode *holeDaten()* bestimmte Datensätze in der MySQL-Datenbank gesucht werden
2. mit der Methode *neuDaten()* bestimmte Datensätze in der MySQL-Datenbank hinzugefügt bzw. geändert werden.

Die beiden Methoden *gibSpaltennamen()* und *gibWerte()* liefern die für die Tabelle in der grafischen Oberfläche benötigten Daten.

Zusätzlich ist in der Klasse *DBAbfrage* die Methode *befrageResults()* implementiert. Da diese jedoch als *private* deklariert ist, kann auf diese von

einer anderen Klasse nicht zugegriffen werden und ist somit auch nicht von außen sichtbar.

Die beiden Klassen *DBAbfrage* und *DBVerbindung*, deren Modelle die Abbildung 4.35 darstellt, werden im Projekt *Zugriff* zusammengefasst. Da dieses Projekt die Grundlage für die oben beschriebenen weiteren Aufgaben darstellt, steht dieses auf dem Schulserver zur Verfügung. In den folgenden Kapiteln, in denen du nun lernen wirst, wie man nach bestimmten Datensätzen sucht, bestimmte Datensätze löscht bzw. ändert, wirst du das Projekt *Zugriff* an die jeweiligen Aufgaben anpassen.

4.7 Datensätze suchen

4.7.1 Die Klasse *DBAbfrage*

Mit Hilfe des Formulars *Datensatz eingeben* hast du bereits einige Datensätze in die Datenbank *Kaffeehaus3* eingefügt. Nun wirst du lernen, wie man mit Hilfe des Formulars *Datensatz suchen* durch Eingabe von Kriterien gezielt nach den Datensätzen bestimmter Kaffees suchen kann.

Bei Suchabfragen wird der Methode *holeDaten()* eine SELECT-Anweisung als Parameter übergeben, die aus Tabelle *Kaffee* diejenigen Datensätze herausucht, die bestimmten Kriterien genügen.

Die SQL-Anweisung

```
SELECT Name, Sorte, Preis, Anzahl, Anbieter  
FROM Kaffee  
WHERE Sorte LIKE 'Bohnenkaffee'
```

sucht aus allen Datensätzen der Tabelle *Kaffee* diejenigen heraus, die im Attribut *Sorte* den Wert „Bohnekaffee“ haben. Diese Abfrage kannst du der Methode *holeDaten()* als Parameter übergeben. Die Klasse *Ansicht* muss aus den Textfeldern die entsprechenden Kriterien herauslesen und diese SQL-Anweisung zusammenstellen.

Übung 4.16:

Speichere das Projekt *Zugriff* unter dem Namen *Suchen01*. Überprüfe, ob die SQL-Anweisung

```
SELECT Name, Sorte, Preis, Anzahl, Anbieter  
FROM Kaffee  
WHERE Sorte LIKE 'Bohnenkaffee'
```

aus der Datenbank *Kaffeehaus03* alle Bohnenkaffees liefert. Vergleiche dazu die Abbildungen 4.36 bzw. 4.37.

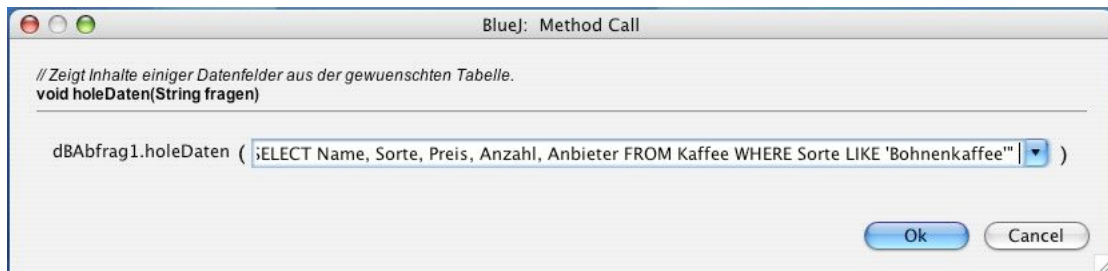


Abbildung 4.36: Aufruf der Methode *holeDaten()* mit dem Parameter "SELECT Name, Sorte, Preis, Anzahl, Anbieter FROM Kaffee WHERE Sorte LIKE 'Bohnenkaffee'"

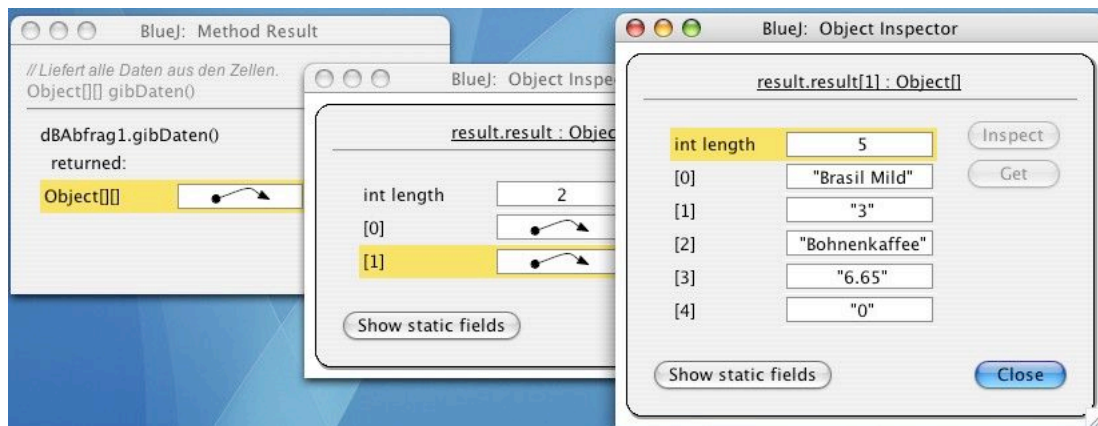



Abbildung 4.37: Die Methode *gibWerte()* liefert alle Bohnenkaffees

Die Klasse *DBAbfrage* kannst du also unverändert übernehmen, auch wenn die Methode *neuDaten()* beim Suchen von Datensätzen momentan keine Verwendung findet.

4.7.2 Die Klasse *Ansicht*

Die Abbildung 4.38 zeigt eine einfache grafische Oberfläche eines solchen Formulars zur Datensuche.



Name	Sorte	Preis	Anzahl	Anbieter
Espresso dOro	Espresso	7.30	0	Dallmayr
Brasilian	Pulverkaffee	4.80	0	Dallmayr
Beste Bohne	Bohnenkaffee	5.30	45	Dallmayr
Mocca Gold	Bohnenkaffee	7.55	0	Dallmayr

Abbildung 4.38: Grafische Benutzeroberfläche des Formulars *Datensatz suchen*

Dieses Formular *Datensatz suchen* unterscheidet sich nur wenig vom Formular *Datensatz eingeben*. Für die Auswahl der Datenfelder (Name, Sorte, usw.) der Tabelle *Kaffee* wird eine ComboBox (Auswahlmenü) verwendet. In das darunter liegende Textfeld werden die Werte eingetragen, die mit den Datenbankeinträgen in der entsprechenden Tabellenspalte verglichen werden. Die Ergebnismenge wird dann in der Tabelle aufgelistet. In der Abbildung 4.38 zeigt die Tabelle alle Kaffees, deren Sorte gleich Bohnenkaffee ist.

Übung 4.17:

Erstelle ein Projekt *Suchen02*, das die in Abbildung 4.38 dargestellte Benutzeroberfläche erzeugt. Du kannst dazu entweder das bereits bekannte Formular *Datensatz eingeben* (vgl. Projekt *Eingabe04b*) oder das auf dem Schulserver liegende Projekt *SuchenGUI* anpassen.

Füge anschließend die beiden Klassen *DBAbfrage* und *DBVerbindung* aus deinem Projekt *Suchen01* hinzu, so dass die Tabelle nach dem Öffnen alle Kaffees auflistet, die in der Tabelle *Kaffee* der Datenbank *Kaffeehaus3* enthalten sind.

Zum Schluss muss die Suchabfrage auf den Button *suchen* reagieren. Hierfür verwendest du die Methode *suchenBActionPerformed()*. Diese muss aus dem Textfeld *sucheSuTF* und der Combobox *attributCB* das Suchkriterium herauslesen und hieraus die SELECT-Anweisung *qryKaffee* zusammenstellen. Dieser String *qryKaffee* wird der Methode *holeKaffee()* als Parameter übergeben. Das Neuzeichnen der Tabelle hast du bereits bei der Erstellung des Eingabeformulars gelernt.

```
/** Reagiert auf Druecken von suchenB. */
private void suchenBActionPerformed(ActionEvent e)
{
    String kriterium = kriteriumSuTF.getText();
    String attribut = (String) attributCB.getSelectedItem();

    String qryKaffee = "SELECT Name, Sorte, Preis, Anzahl, Anbieter " +
        "FROM Kaffee " +
        "WHERE " + attribut + " LIKE '" + kriterium + "'";

    tabellenP.remove(scrollPane);
    holeMySQLDaten(qryKaffee);
    tabellenP.add(scrollPane);
    pack();
}
```

Abbildung 4.39: Die Methode *suchenBActionPerformed()* in der Klasse *Ansicht*

Übung 4.18:

Speichere dein Projekt *Suchen02* unter dem Namen *Suchen03*. Implementiere die Methode *suchenBActionPerformed()*. Du solltest ein ähnliches Ergebnis erhalten wie in *Abbildung 4.38*.

Bemerkung:

Die *Abbildung 4.38* zeigt zusätzlich den Button *tabelle*. Wie bereits im Projekt *Eingabe04* beschrieben, kann mit Hilfe dieses Buttons die komplette Tabelle *Kaffee* aus der Datenbank *Kaffeehaus2* wieder dargestellt werden.

4.8 Datensätze löschen

4.8.1 Die Klasse *DBAbfrage*

Nun wirst du lernen, wie man aus der Tabelle *Kaffee* den Datensatz eines bestimmten Kaffees entfernen kann. Das Modell der Klasse *DBAbfrage* für die Eingabe von Datensätzen kann für das Löschen von Datensätzen unverändert übernommen werden

Zum Entfernen von Datensätzen wird der Methode *neuDaten()* eine DELETE-Anweisung als Parameter übergeben, die aus der Tabelle *Kaffee* denjenigen Datensatz entfernt, der einem bestimmten Kriterium genügt.

Die SQL-Anweisung

```
DELETE FROM Kaffee
WHERE Name LIKE 'Classico'
```

sucht aus allen Datensätzen der Tabelle *Kaffee* denjenigen heraus, der den Namen „Classico“ hat und entfernt diesen.

Voraussetzung für die Methode *neuDaten()* ist, dass in der Tabelle *Kaffee* zu jedem Namen auch nur ein einziger Kaffee gehört. Somit ist jeder Kaffee durch seinen Namen eindeutig identifizierbar.

Bemerkung:

Die Datenbank *Kaffeehaus3* verwendet zur eindeutigen Identifizierung der Datensätze den Primärschlüssel *KaffeeID*. Dieser wird jedoch in der Tabelle der GUI nicht mit aufgelistet und steht somit dem Benutzer als Kriterium auch nicht zur Verfügung. Man müsste eine Möglichkeit finden, aus dem Eintrag in der Tabelle der GUI eindeutig auf diesen Primärschlüssel zu schließen. Die Verwendung des Namens ist dann sinnvoll, wenn dieser Kaffeename nur einmal existiert.

Übung 4.19:

Speichere das Projekt *Zugriff* unter dem Namen *Loeschen01*. Überprüfe, ob die SQL-Anweisung

```
DELETE FROM Kaffee  
WHERE Name LIKE 'Mocca Gold'
```

aus der Datenbank *Kaffeehaus03* den entsprechenden Kaffee entfernt. Vergleiche dazu die Abbildungen 4.40 bis 4.43

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
4	Gran Gusto	Espresso	5.60	25	Cellini
5	Cafe Range	Schonkaffee	8.35	60	Hochland
6	Beste Bohne	Bohnenkaffee	5.30	45	Dallmayr
7	Feine Milde	Schonkaffee	7.25	0	Hochland
8	Mocca Gold	Bohnenkaffe	7.55	0	Dallmayr
9	Cafe Picco	Espresso	6.95	0	Hochland

Abbildung 4.40: Die Datenbank *Kaffeehaus3* vor dem Löschen von „Mocca Gold“

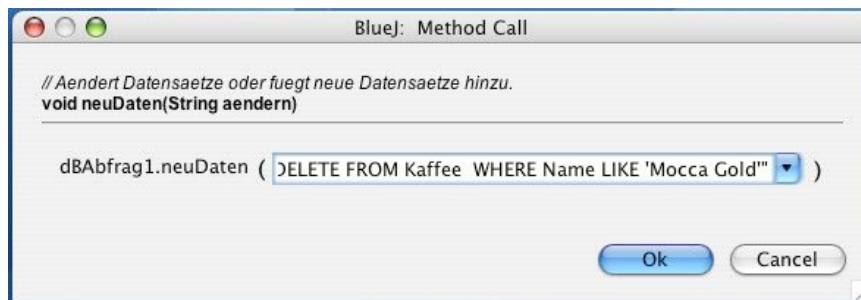


Abbildung 4.41: Aufruf der Methode *neuDaten()* mit dem Parameter "DELETE FROM Kaffee WHERE Name LIKE 'Mocca Gold'"

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
4	Gran Gusto	Espresso	5.60	25	Cellini
5	Cafe Range	Schonkaffee	8.35	60	Hochland
6	Beste Bohne	Bohnenkaffee	5.30	45	Dallmayr
7	Feine Milde	Schonkaffee	7.25	0	Hochland
9	Cafe Picco	Espresso	6.95	0	Hochland

Abbildung 4.42: Die Datenbank *Kaffeehaus2* nach dem Löschen „Mocca Gold“

Die Klasse *DBAbfrage* kannst du also unverändert übernehmen, auch wenn die Methode *holeDaten()* beim Löschen von Datensätzen momentan keine Verwendung findet.

4.8.2 Die Klasse *Ansicht*

Die Abbildung 4.43 zeigt eine einfache grafische Oberfläche des Formulars *Datensatz loeschen*. Es unterscheidet sich kaum vom Formular *Datensatz suchen*.

Übung 4.20:

Erstelle ein Projekt *Loeschen02*, das die in Abbildung 4.43 dargestellte Benutzeroberfläche erzeugt. Du kannst dazu entweder das bereits bekannte Formular *Datensatz eingeben* (vgl. Projekt *Eingabe04b*) oder das auf dem Schulserver liegende Projekt *LoeschenGUI* anpassen.

Füge anschließend die beiden Klassen *DBAbfrage* und *DBVerbindung* aus deinem Projekt *Loeschen01* hinzu, so dass die Tabelle nach dem Öffnen alle Kaffees auflistet, die in der Tabelle *Kaffee* der Datenbank *Kaffeehaus3* enthalten sind.

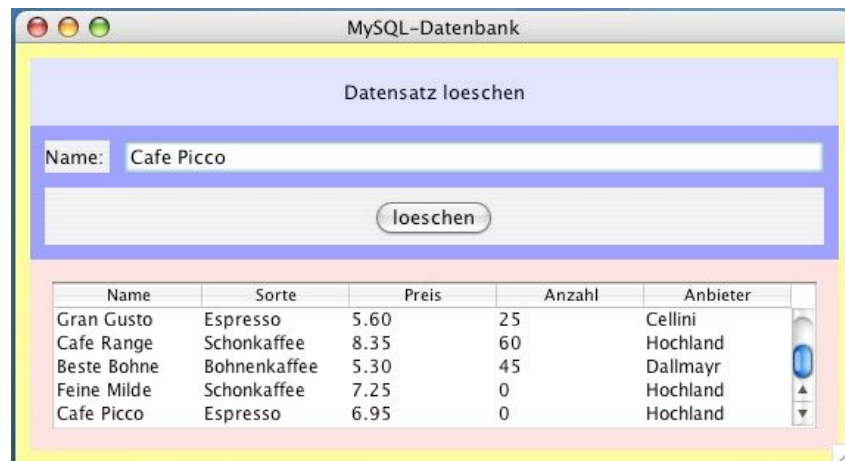


Abbildung 4.43: Grafische Benutzeroberfläche des Formulars *Datensatz loeschen*

Zum Schluss muss die Suchabfrage auf den Button *loeschen* reagieren. Hierfür verwendest du die Methode *loeschenBActionPerformed()*. Diese liest aus dem *nameLoTF* den Namen des zu löschenden Kaffees heraus und stellt die DELETE-Anweisung *delKaffee* zusammen. Dieser String *delKaffee* wird der Methode *neuDaten()* als Parameter übergeben. Das Neuzeichnen der Tabelle hast du bereits bei der Erstellung des Formulars *Datensatz eingeben* gelernt.

```

/** Reagiert auf Druecken von einfuegenB. */
private void loeschenBActionPerformed(ActionEvent e)
{
    //holt sich den Kaffeenamen aus dem Textfeld
    String name = nameLoTF.getText();
    //holt sich den Kaffeenamen aus der gewählten Tabellenreihe
    //int reihe = tabelle.getSelectedRow();
    //Object name = tabelle.getValueAt(reihe, 0);

    String delKaffee = "DELETE FROM Kaffee " +
        "WHERE Name LIKE '" + name + "'";
    String qryKaffee = "SELECT Name, Sorte, Preis, Anzahl, Anbieter " +
        "FROM Kaffee";

    dbAbfrage = new DBAbfrage();
    dbAbfrage.neuDaten(delKaffee);

    tabellenP.remove(scrollPane);
    holeMySQLDaten(qryKaffee);
    tabellenP.add(scrollPane);
    pack();
}

```

Abbildung 4.44: Die Methode *loeschenBActionPerformed()* in der Klasse *DBAbfrage*

Bemerkung:

Der Name des gesuchten Kaffees kann auch durch Anklicken der entsprechenden Reihe in der Tabelle der GUI ausgewählt werden. Ich habe diese Möglichkeit in der Methode *loeschenBActionPerformed()* als Kommentar angegeben.

Übung 4.21:

Speichere dein Projekt *Loeschen02* unter dem Namen *Loeschen03*. Implementiere die Methode *loeschenBActionPerformed()*. Du solltest ein ähnliches Ergebnis erhalten wie in Abbildung 4.43. Nach Drücken von *loeschen* fehlt der entsprechende Datensatz in der Tabelle der GUI.

4.9 Datensätze ändern

4.9.1 Die Klasse *DBAbfrage*

Nun wirst du lernen, wie man in der Tabelle *Kaffee* den Datensatz eines bestimmten Kaffees ändern kann. Das Modell der Klasse *DBAbfrage* für die Eingabe von Datensätzen kann auch für das Ändern von Datensätzen fast unverändert übernommen werden.

Zum Ändern von Datensätzen wird der Methode *neuKaffee()* eine UPDATE-Anweisung als Parameter übergeben, die in der Tabelle *Kaffee* denjenigen Datensatz ändert, dessen Namen einem bestimmten Kriterium genügt.

Bemerkung

Die Verwendung des Namens zur eindeutigen Identifizierung des Datensatzes ist dann sinnvoll, wenn dieser Kaffee Name nur einmal existiert. Da in der grafischen Benutzeroberfläche der Datensatz aus der Tabelle der GUI gewählt wird, könnte man hiermit aus der Datenbank den Primärschlüssel *KaffeeID* ermitteln und somit auch verwenden.

Die SQL-Anweisung

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

sucht aus allen Datensätzen der Tabelle *Kaffee* denjenigen Kaffee heraus, dessen Namen „Espresso dOro“ ist und ändert den entsprechenden Wert.

Das gleichzeitige Ändern mehrerer Datenfelder eines Datensatzes erfolgt mittels Transaktionen, die du bereits Kapitel 3.3 *Verwendung von Transaktionen* gelernt hast.

Im Projekt *Aendern01* wirst du nun die Werte in den beiden Datenfelder *Preis* und *Anzahl* verändern. Hierzu werden zwei UPDATE-Anweisungen benötigt. Deswegen musst du nun eine weitere Methode *aendereDaten()* in der Klasse *DBAbfrage* implementieren, die eine Transaktion durchführen kann. Das erweiterte Modell der Klasse *DBAbfrage* zeigt Abbildung 4.45.



Abbildung 4.45: Erweitertes Modell der Klasse *DBAbfrage*

Den vorläufigen Quelltext der Methode *aendereDaten()* zeigt Abbildung 4.46.

```

/** Aendert einen Datensatz in der gewuenschten Tabelle. */
public void aendereDaten(String updPreis, String updAnzahl)
{
    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 4.46: Die Methode *aendereDaten()* in der Klasse *DbAbfrage*

Die Methode *aendereDaten()* erwartet zwei UPDATE-Anweisungen als String. Für die Transaktion wird zusätzlich das Connection-Objekt *con* benötigt, das von der Klasse *DBVerbindung* geliefert wird.

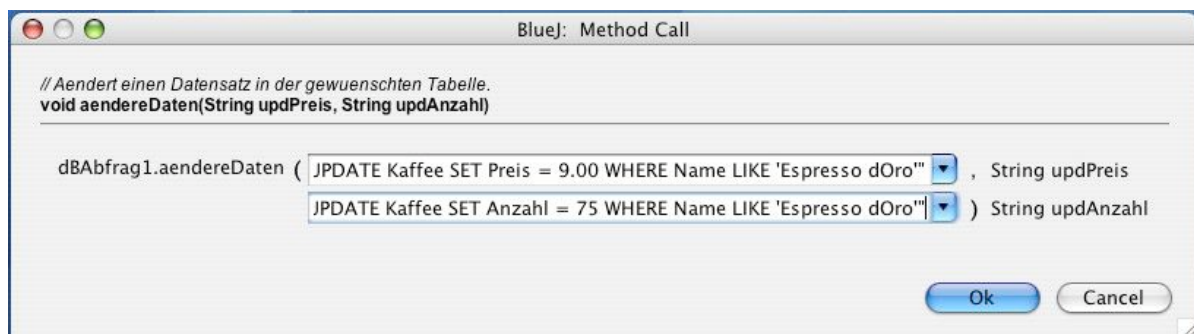
Übung 4.22:

Speichere das Projekt *Zugriff* unter dem Namen *Aendern01*. Implementiere die Methode *aendereDaten()* in der Klasse *DBAbfrage*. Überprüfe, ob die SQL-Anweisungen

```
"UPDATE Kaffee SET Preis = 9.00 WHERE Name LIKE 'Espresso dOro'"
"UPDATE Kaffee SET Anzahl = 75 WHERE Name LIKE 'Espresso dOro'"
```

in der Datenbank *Kaffeehaus3* den Datensatz des entsprechenden Kaffees ändern. Vergleiche dazu die Abbildungen 4.47 bis 4.49.

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
4	Gran Gusto	Espresso	5.60	25	Cellini
5	Cafe Range	Schonkaffee	8.35	60	Hochland
6	Beste Bohne	Bohnenkaffee	5.30	45	Dallmayr
7	Feine Milde	Schonkaffee	7.25	0	Hochland

Abbildung 4.47: Die Datenbank *Kaffeehaus3* vor dem Ändern von „Espresso dOro“**Abbildung 4.48:** Aufruf der Methode *aendereDaten()* mit dem Parameter "UPDATE Kaffee SET Preis = 9.00 WHERE Name LIKE 'Espresso dOro'" "UPDATE Kaffee SET Anzahl = 75 WHERE Name LIKE 'Espresso dOro'"

KaffeeID	Name	Sorte	Preis	Anzahl	Anbieter
1	Espresso dOro	Espresso	9.00	75	Dallmayr
2	Brasil Mild	Bohnenkaffee	6.40	50	Tchibo
3	Brasilian	Pulverkaffee	4.80	0	Dallmayr
4	Gran Gusto	Espresso	5.60	25	Cellini
5	Cafe Range	Schonkaffee	8.35	60	Hochland
6	Beste Bohne	Bohnenkaffee	5.30	45	Dallmayr
7	Feine Milde	Schonkaffee	7.25	0	Hochland

Abbildung 4.49: Die Datenbank *Kaffeehaus3* nach dem Ändern von „Espresso dOro“

4.9.2 Die Klasse *Ansicht*

Die Abbildung 3.50 zeigt eine einfache grafische Oberfläche des Formulars *Datensatz aendern*. Es unterscheidet sich kaum vom Formular *Datensatz eingeben*.

Übung 4.23:

Erstelle ein Projekt *Aendern02*, das die in Abbildung 4.50 dargestellte Benutzeroberfläche erzeugt. Du kannst dazu entweder das bereits bekannte Formular *Datensatz eingeben* (vgl. Projekt *Eingabe04b*) oder das auf dem Schulserver liegende Projekt *AendernGUI* anpassen.

Füge anschließend die beiden Klassen *DBAbfrage* und *DBVerbindung* aus deinem Projekt *Aendern01* hinzu, so dass die Tabelle nach dem Öffnen alle Kaffees auflistet, die in der Tabelle *Kaffee* der Datenbank *Kaffeehaus3* enthalten sind.

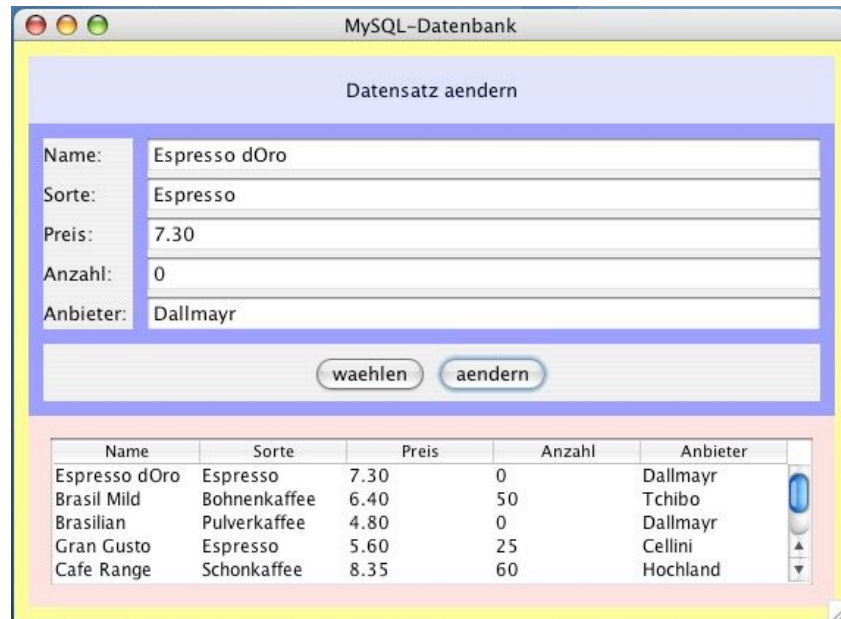


Abbildung 4.50: Grafische Benutzeroberfläche des Formulars *Datensatz aendern*

Der Button *wählen* holt einen in der Tabelle der GUI ausgewählten Datensatz in die entsprechenden Textfelder. Diese Aufgabe erledigt die Methode *wählenBActionPerformed()*:

```
/** Reagiert auf Druecken von waehlenB. */
private void waehlenBActionPerformed(ActionEvent e)
{
    //holt sich den Kaffeedaten aus der gewählten Tabellenreihe
    int reihe = tabelle.getSelectedRow();
    String name = (String)tabelle.getValueAt(reihe, 0);
    String sorte = (String)tabelle.getValueAt(reihe, 1);
    String preis = (String)tabelle.getValueAt(reihe, 2);
    String anzahl = (String)tabelle.getValueAt(reihe, 3);
    String anbieter = (String)tabelle.getValueAt(reihe, 4);
    //schreibt den gewählten Datensatz in die Textfelder
    nameAenTF.setText(name);
    sorteAenTF.setText(sorte);
    preisAenTF.setText(preis);
    anzahlAenTF.setText(anzahl);
    anbieterAenTF.setText(anbieter);
}
```

Abbildung 4.51: Die Methode *wählenBActionPerformed()* in der Klasse *Ansicht*

Übung 4.24

Speichere dein Projekt *Aendernn02* unter dem Namen *Aendern03*. Implementiere die Methode *wählenBActionPerformed()*. Du solltest ein

ähnliches Ergebnis erhalten wie in Abbildung 4.50. Nach dem Auswählen eines Datensatzes in der Tabelle der GUI und dem anschließenden Drücken von *wählen* wird der entsprechende Datensatz in den Textfeldern angezeigt.

Der Benutzer kann nun in allen Textfeldern außer dem *namenAenTF* Änderungen durchführen. Das *namenAenTF* wird mit *setEditable(false)* als nichteditierbar gesetzt, da dieses Datenfeld zur Identifizierung des Datensatzes verwendet wird.

Zum Schluss muss die Änderung des Datensatzes mit ein Klick auf den Button *aendern* fixiert werden. Hierfür verwendest du die Methode *aendernBActionPerformed()*. Diese liest aus den Textfeldern die Werte heraus und stellt die UPDATE-Anweisungen *updSorte*, *updPreis*, *updAnzahl* und *updAnbieter* zusammen. Diese Strings werden der Methode *aendereDaten()* als Parameter übergeben. Das Neuzeichnen der Tabelle hast du bereits bei der Erstellung des Eingabeformulars gelernt.

```
/** Reagiert auf Druecken von aendernB. */
private void aendernBActionPerformed(ActionEvent e)
{
    String name = nameAenTF.getText();
    String sorte = sorteAenTF.getText();
    double preis = Double.parseDouble(preisAenTF.getText());
    int anzahl = Integer.parseInt(anzahlAenTF.getText());
    String anbieter = anbieterAenTF.getText();

    String updSorte = "UPDATE Kaffee " +
        "SET Sorte = '" + sorte + "' " +
        "WHERE Name LIKE '" + name + "'";
    String updPreis = "UPDATE Kaffee " +
        "SET Preis = " + preis + " " +
        "WHERE Name LIKE '" + name + "'";
    String updAnzahl = "UPDATE Kaffee " +
        "SET Anzahl = " + anzahl + " " +
        "WHERE Name LIKE '" + name + "'";
    String updAnbieter = "UPDATE Kaffee " +
        "SET Anbieter = '" + anbieter + "' " +
        "WHERE Name LIKE '" + name + "'";

    String qryKaffee = "SELECT Name, Sorte, Preis, Anzahl, Anbieter " +
        "FROM Kaffee";

    dbAbfrage = new DBAbfrage();
    dbAbfrage.aendereDaten(updSorte, updPreis, updAnzahl, updAnbieter);

    tabellenP.remove(scrollPane);
```

```

holeMySQLDaten(qryKaffee);
tabellenP.add(scrollPane);
pack();
}

```

Abbildung 4.52: Die Methode *aendernBActionPerformed()* in der Klasse *Ansicht*

Übung 4.25:

Implementiere in deinem Projekt *Aendern03* in der Klasse *Ansicht* die Methode *aendernBActionPerformed()*. Vergiss nicht entsprechende Änderungen der Methode *aendereDaten()* in der Klasse *DBAbfrage*. Du solltest ein ähnliches Ergebnis erhalten wie in Abbildung 4.50. Nach Drücken von *aendern* wird der geänderte Datensatz in der Tabelle der GUI angezeigt.

4.10 Der Kaffeeladen

Im Projekt *Kaffeeladen* werden die vier Formulare

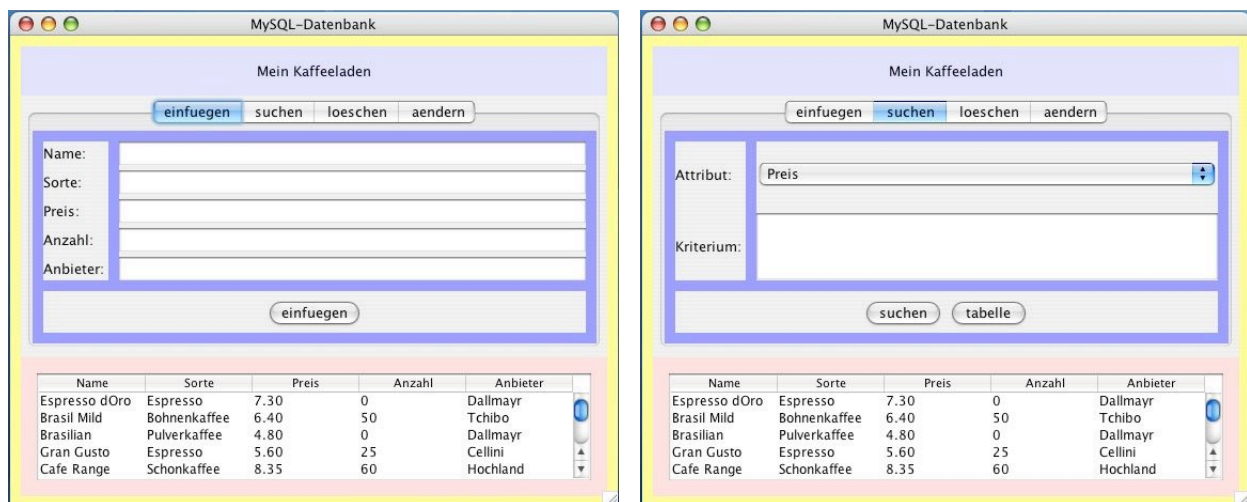
Datensatz eingeben

Datensatz suchen

Datensatz loeschen

Datensatz aendern

als Registerkarten in der grafischen Oberfläche dargestellt.



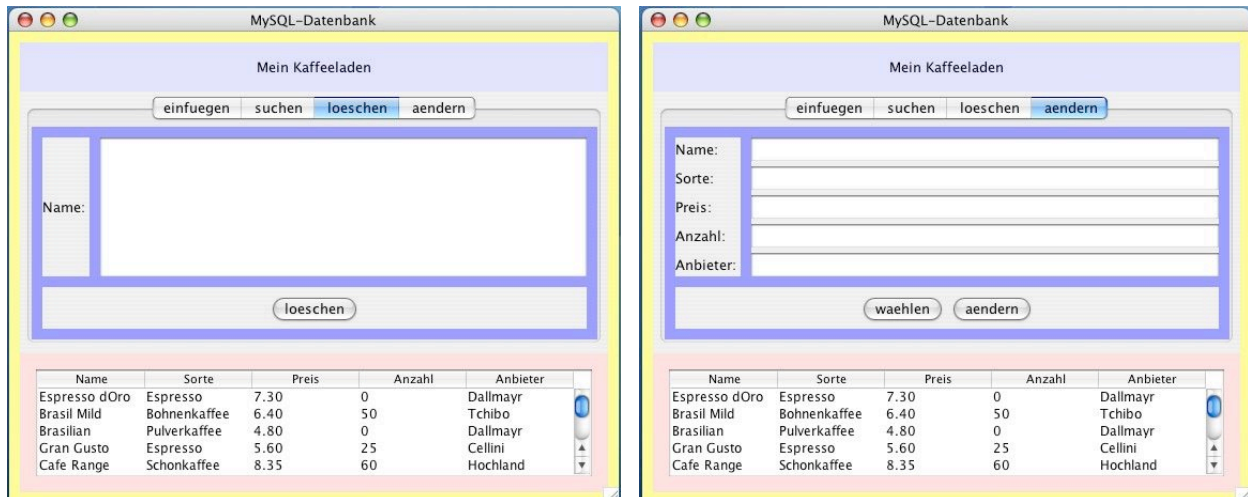


Abbildung 4.53: Grafische Oberfläche mit den vier Registerkarten

Dieses Projekt lässt sich weitgehend mit Kopieren und Einfügen der bereits bekannten Formulare erstellen.

Bemerkung:

Die Textfelder in den Registerkarten *suchen* und *loeschen* werden nicht in der geeigneten Größe dargestellt. Diese kannst du später mit Hilfe der Methoden der entsprechenden Layoutmanager anpassen.

In der Methode *erzeugeFenster()* wird auf das *mainP* anstelle der bisherigen Formularpanels ein *tabP* gesetzt

```
private void erzeugeFenster()
{
    ..... weitere Anweisungen .....

    //Entsprechende Panels werden erstellt.
    mainP.add(namenPERstellen());
    mainP.add(tabPERstellen());
    mainP.add(tabellenPERstellen());

    ..... weitere Anweisungen .....
}
```

Abbildung 4.54: Die Methode *erzeugeFenster()* in der Klasse *Ansicht*

Zur Erstellung von Registerkarten verwendest du die Klasse *JTabbedPane*. Dem Panel *tabP* werden der Reihe nach alle Registerkarten hinzugefügt.

```
private Container tabPERstellen()
{
    tabP = new JTabbedPane();
    tabP.add("einfuegen", einfuegenPERstellen());
    tabP.add("suchen", suchenPERstellen());
    tabP.add("loeschen", loeschenPERstellen());
    tabP.add("aendern", aendernPERstellen());
    return tabP;
}
```

Abbildung 4.55: Die Methode *tabPERstellen()* in der Klasse *Ansicht*

Übung 4.26:

Erstelle das Projekt *Kaffeeladen*. Falls du in den bisherigen Übungen die gleichen Namen für die Datenfelder verwendet hast, solltest du ohne größere Probleme nun mit *Copy and Paste* bzw. *Add Class from File ...* alle benötigten Methoden und Klassen zusammenstellen können.

In diesem Kapitel hast du nun eine einfache Datenbank *Kaffeehaus3* verwendet. Diese bestand aus einer einzigen Tabelle *Kaffee*.

In einer Java-Anwendung kannst du nun

- a) Datensätze eingeben,
- b) Datensätzen suchen,
- c) Datensätze löschen und
- d) Datensätze ändern.

Allerdings möchte ich zum Schluss noch einmal erwähnen, dass diese Datenbank nicht den Normalformen genügt.