

5 Kaffeehandel

Wie bereits erwähnt, entspricht die Datenbank Kaffeehaus nicht den Normalformen. Die Datenfelder *Sorte* und *Anbieter* müssen in eine separate Tabelle ausgelagert werden. Zusätzlich soll der *Geschmack* der einzelnen Kaffees verwaltet werden. Abbildung 5.1 zeigt das zugehörige Klassendiagramm der Datenbank *Kaffeehandel4*, das bereits in Kapitel 2 vorgestellt wurde.

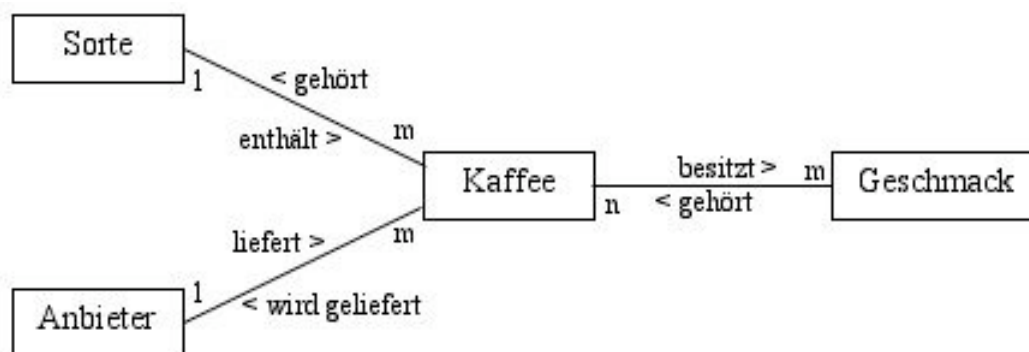


Abbildung 5.1: Klassendiagramm von *Kaffeehandel4*

Zwischen den einzelnen Klassen bestehen also folgende Beziehungen:

Eine bestimmter Anbieter liefert **m** (eine oder mehrere) Kaffees.

Eine bestimmter Kaffee wird geliefert von **1** (genau einem) Anbieter.

Eine bestimmte Dorte enthält **m** (eine oder mehrere) Kaffees.

Eine bestimmter Kaffee gehört zu **1** (genau einer) Sorte.

Eine bestimmter Kaffee besitzt **m** (eine oder mehrere) Geschmacksrichtungen.

Eine bestimmte Geschmacksrichtung gehört zu **n** (einem oder mehreren) Kaffees.

In einer relationalen Datenbank kannst du die beiden 1:m-Beziehungen mit der Einführung von Primärschlüsseln in den Klassen *Sorte* und *Anbieter* und den zugehörigen Fremdschlüsseln in der Klasse *Kaffee* realisieren. Die n:m-Beziehung zwischen den Klassen *Kaffee* und *Geschmack* musst du in einer zusätzlichen Zuordnungstabelle *Kaff_Geschm* erfassen.

Übung 5.1:

Implementiere die Datenbank *Kaffeehandel4*. Verwende hierzu die in Kapitel 2 beschriebenen Grundlagen in MySQL und phpMyAdmin. Abbildung 5.2 zeigt

die Klassen mit ihren jeweiligen Datenfeldern. Ich empfehle, die gleichen Namen der Datenfelder zu übernehmen, damit du die entsprechenden SQL-Anweisungen aus diesem Skript übernehmen kannst.

Feld	Typ
<u>SorteID</u>	Int(11)
Sorte	varchar(100)

Feld	Typ
<u>KaffeeID</u>	Int(11)
Name	varchar(100)
SorteNr	Int(11)
Preis	decimal(3,2)
Anzahl	Int(11)
AnbieterNr	Int(11)

Feld	Typ
<u>AnbieterID</u>	Int(11)
Name	varchar(100)
PLZ	varchar(5)
Ort	varchar(100)

Feld	Typ
<u>KaffeeNr</u>	Int(11)
<u>GeschmackNr</u>	Int(11)

Feld	Typ
<u>GeschmackID</u>	Int(11)
Art	varchar(100)

Abbildung 5.2: Die Klassen *Anbieter*, *Sorte*, *Kaffee*, *Kaff_Geschm* und *Geschmack*

Die Datenbank *Kaffeehandel4* soll nun mit Hilfe einer geeigneten grafischen Benutzeroberfläche gefüllt werden. Allerdings müssen die 1:m-Beziehung und die n:m-Beziehung auf verschiedene Weise in dieser Benutzeroberfläche realisiert werden.

5.1 Implementierung der 1:m-Beziehung

Eine Möglichkeit, die 1:m-Beziehung in einer grafischen Benutzeroberfläche zu realisieren, ist die Verwendung einer Combobox *JComboBox*. Abbildung 5.3 zeigt eine mögliche grafische Benutzeroberfläche. Eine solche Combobox bietet dem Benutzer alle Sorten bzw. alle Anbieter zur Auswahl an. Die Inhalte der Comboboxen müssen aus den Tabellen *Sorte* bzw. *Anbieter* entnommen werden. Zum jetzigen Zeitpunkt sind diese beiden Tabellen allerdings noch leer. Deswegen werden in das *namenP* die beiden Buttons *Neue Sorte* und *Neuer Anbieter* eingesetzt, die ein separates Eingabefenster zum Auffüllen der jeweiligen Tabellen öffnen.

5.1.1 Eingabeformulare für neue Sorten und Anbieter

Im ersten Schritt wirst du lernen, wie die beiden Buttons *Neue Sorte* und *Neuer Anbieter* zwei separate Eingabefenster öffnen, um mit den eingegebenen Daten die Tabellen *Sorte* und *Anbieter* der Datenbank *Kaffeehandel4* zu füllen.

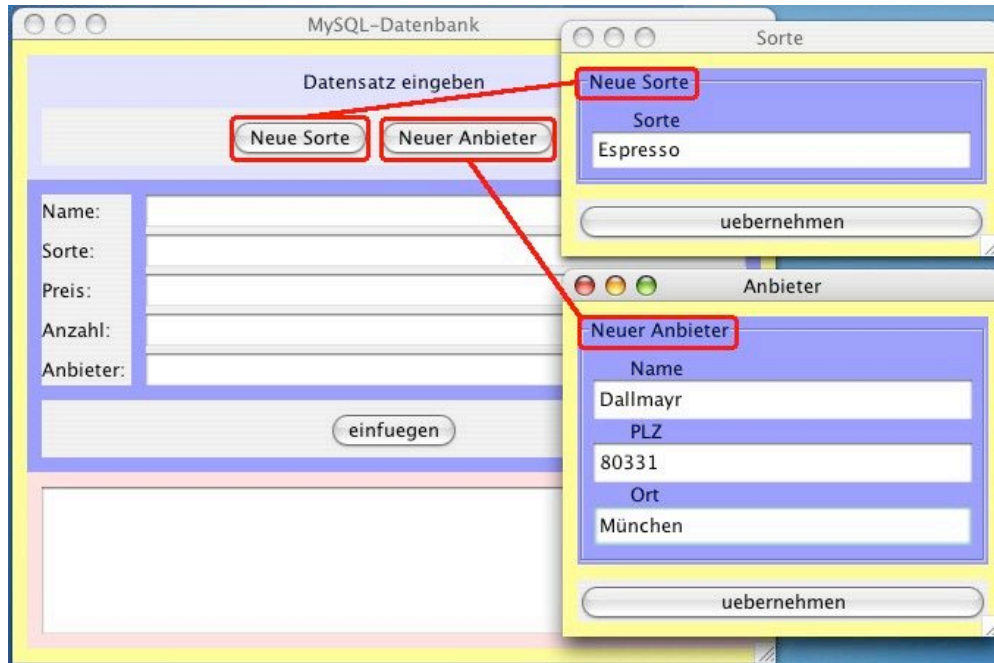


Abbildung 5.3: Die Buttons öffnen die entsprechenden Eingabeformulare

Übung 5.2:

- Hole vom Schulserver das Projekt *HandelGUI* und speichere dieses als Projekt *Handel01*. Implementiere die fehlenden Buttons *Neue Sorte* und *Neuer Anbieter*, sowie die Labels und Textfelder für Preis, Anzahl und Anbieter.
- Hole vom Schulserver das Projekt *SorteGUI*. Füge hieraus mit *Edit > Add Class from File...* die Klasse *Sorte* dem Projekt *Handel01* hinzu.
- Erstelle weitere Klasse *Anbieter* mit den Textfeldern Name, PLZ und Ort.
- Füge mit *Edit > Add Class from File...* die Klassen *DBVerbindung* und *DBAbfrage* dem Projekt *Zugriff* (Kapitel 4) hinzu.

Nun sollte das Projekt *Handel01* das in Abbildung 5.4 dargestellte Klassendiagramm zeigen.

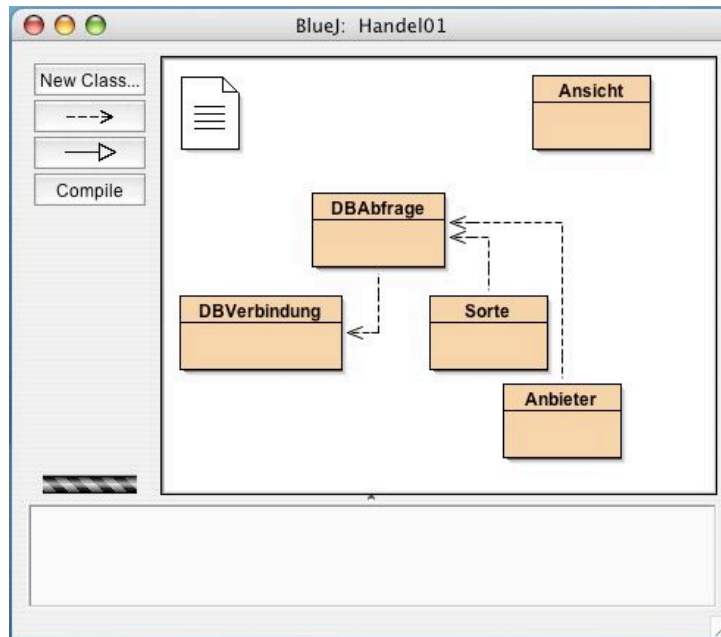


Abbildung 5.4: Klassendiagramm des Projekts *Handel01*

Drückt der Benutzer auf den Button *Neue Sorte*, so öffnet sich das Eingabeformular für eine neue Sorte. Diese Aktion wird in der Methode *sortNeuBActionPerformed()* implementiert.

```
/** Reagiert auf Druecken von sortNeuB. */
private void sortNeuBActionPerformed(ActionEvent e)
{
    JFrame frame = new Sorte();
}
```

Abbildung 5.5: Die Methode *sorteNeuBActionPerformed()* in der Klasse *Ansicht*

Mit einem Klick auf *übernehmen* wird der Text aus dem *sorteTF* gelesen und in die Tabelle *Sorte* der Datenbank *Kaffeeladen4* eingelesen. Darauf wird das Fenster geschlossen.

```
/** Reagiert auf Druecken von uebernehmenB. */
private void uebernehmenBActionPerformed(ActionEvent e)
{
    String sorte = sorteTF.getText();
    String updSorte = "INSERT INTO Sorte(sorte) " +
        "VALUES ('" + sorte + "')";

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

    setVisible(false);
}
```

```
dispose();
}
```

Abbildung 5.6: Die Methode *uebernehmenBActionPerformed()* in der Klasse *Sorte*

Übung: 5.3:

- Implementiere die beiden Methoden *sortNeuBActionPerformed()* und *uebernehmenBActionPerformed()*.
- Implementiere entsprechende Methoden für die Eingabe neuer Anbieter.
(vgl. Projekt *Handel01a*)



Abbildung 5.7: Die vom Benutzer eingegebenen Daten werden in die Tabellen übernommen

5.1.2 Comboboxen für Sorte und Anbieter

Im zweiten Schritt lernst, wie die Daten aus den Tabellen *Sorte* und *Anbieter* der Datenbank *Kaffeehandel04* in die Comboboxen übernommen werden.

In der folgenden Übung wirst du das Textfeld *sorteEinTF* ersetzen durch die Combobox *sorteCB*.

Übung 5.4:

- Im Katalog der Datenfelder entferne *sorteEinTF* und schreibe dafür:
`private JTextField nameEinTF, preisEinTF, anzahlEinTF, anbieterEinTF;`

```
private JComboBox sorteCB;
```

2. In der Methode *ein fuegenPERstellen()* entferne alle Anweisungen zur Darstellung von *sorteEinTF* und schreibe dafür:

```
String[] sortenamen = {"Espresso", "Bohnenkaffee", "Schonkaffee"};  
sorteCB = new JComboBox(sortenamen);  
sorteCB.setSelectedIndex(2);  
textEinP.add(sorteCB);
```

Teste nun dein Projekt *Handel01*. Welches Problem tritt nun?

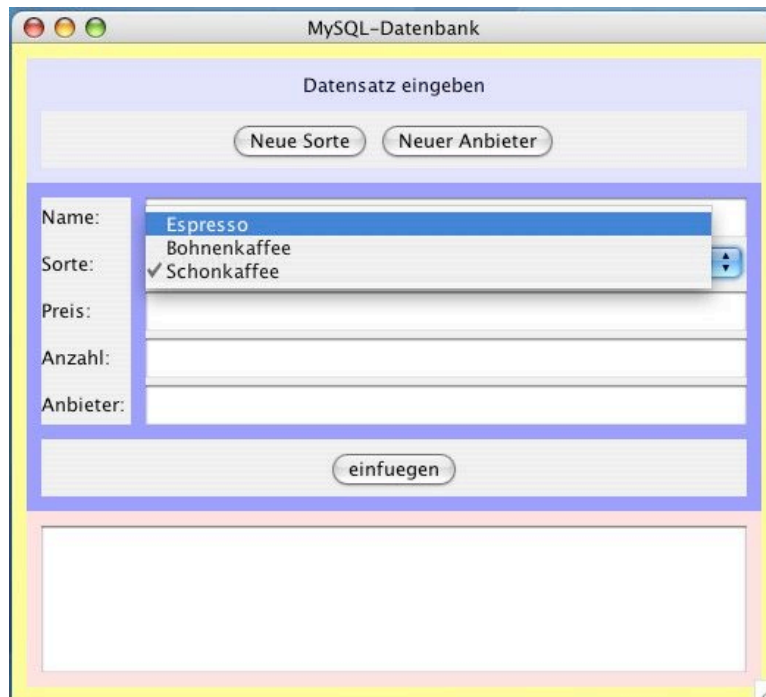


Abbildung 5.8: Die Sorten werden in einer Combobox dargestellt

Die grafische Benutzeroberfläche zeigt eine Combobox wie in Abbildung 5.8. Du kannst auch neue Sorten in die Datenbank einfügen. Allerdings werden diese neuen Sorten nicht in der Combobox aufgelistet. Um die Werte in der Combobox während der Laufzeit des Programms zu aktualisieren, müssen folgende beiden Probleme gelöst werden:

1. Die Namen der Sorten müssen direkt aus der Tabelle *Sorte* entnommen werden.
2. Zur Aktualisierung während der Laufzeit benötigst du ein so genanntes `ComboBoxModel`.

Zuerst wird ein Objekt *sorteModel* in der Liste aller Datenfelder deklariert:

```
private DefaultComboBoxModel sorteModel;
```

Anschließend füllst du in der Methode *holeSortenamen()* dieses *sorteModel* mit den entsprechenden Daten aus der Tabelle *Sorte*.

```
/** Holt sich alle Sortenamen aus der Tabelle Sorte. */
private void holeSortenamen()
{
    String qrySorte = "SELECT Sorte FROM Sorte";

    dbAbfrage = new DBAbfrage();
    dbAbfrage.holeDaten(qrySorte);
    Object[][] werte = dbAbfrage.gibWerte();

    int anzahl = werte.length;
    for (int i = 0; i < anzahl; i++) {
        sorteModel.addElement((String) werte[i][0]);
    }
}
```

Abbildung 5.9: Die Methode *holeSortenamen()* in der Klasse *Ansicht*

Die SELECT-Anweisung *qrySorte* liefert alle Sortenamen aus der Tabelle *Sorte*. Diese werden nun in einer Schleife aus dem zweidimensionalen Array *werte* in das *ComboBoxModel* *sorteModel* übertragen.

Nun setzt du innerhalb der Methode *ein fuegenPErstellen()* die Combobox *sorteCB* zusammen mit seinem Modell *sorteModel* auf das *ein fuegenP*.

..... weitere Anweisungen

```
sorteModel = new DefaultComboBoxModel();
holeSortenamen();
sorteCB = new JComboBox(sorteModel);
textEinP.add(sorteCB);
```

..... weitere Anweisungen

Abbildung 5.10: In der Klasse *Ansicht* wird *anbieterCB* auf das *ein fuegenP* gesetzt

Nun hast du bereits beim Öffnen des Fensters alle Werte der Tabelle *Sorte* in der Combobox zur Verfügung und kannst hieraus eine Sorte auswählen. Im letzten Schritt musst du noch dafür sorgen, dass die Combobox immer wieder nach der Eingabe neuer Sorten wieder aktualisiert wird. Hierzu verwendest du einen dritten Button *aktualisieren* mit der zugehörigen Methode *aktualisierenBtActionPerformed()*.

```
/** Reagiert auf Druecken von aktualisierenB. */  
private void aktualisierenBActionPerformed(ActionEvent e)  
{  
    sorteModel.removeAllElements();  
    holeSortenamen();  
}
```

Abbildung 5.11: Der Button *Anbieter aktualisieren* reagiert nun

Übung 5.5

Implementiere in deinem Projekt *Handel01* in der Klasse *Ansicht* den Button *aktualisieren* und die Methoden, die in den Abbildungen 5.9 bis 5.11 dargestellt werden.

(vgl. Projekt *Handel01b*)

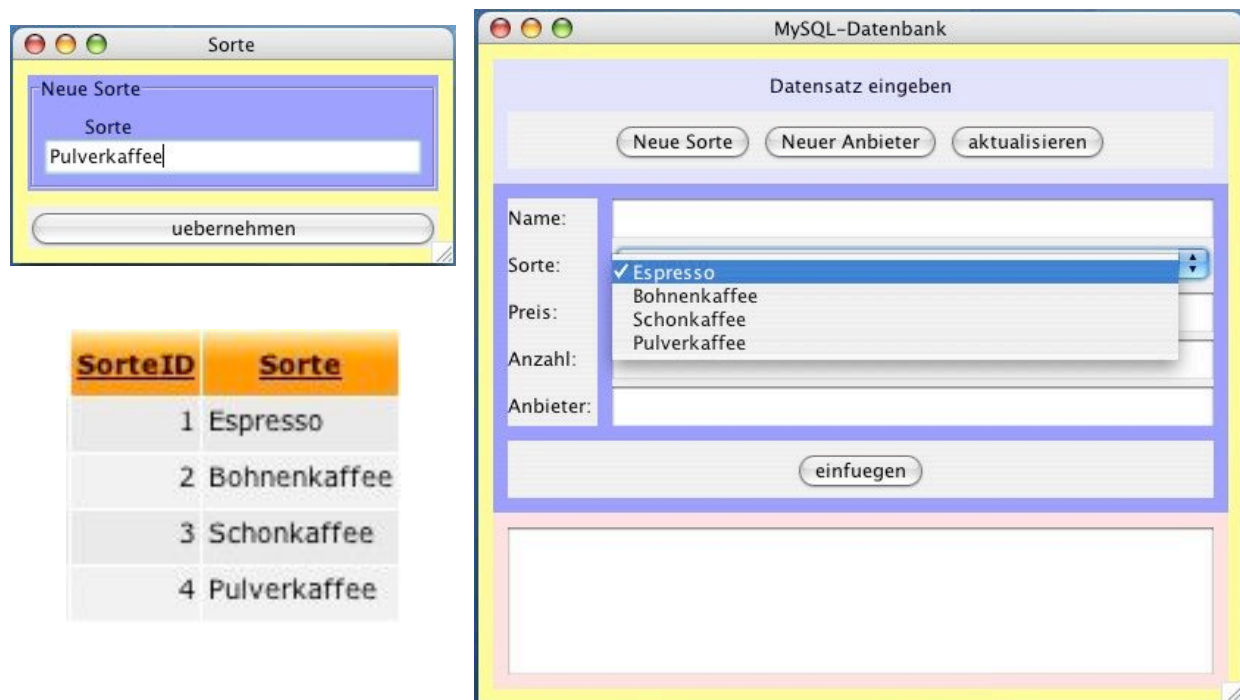


Abbildung 5.12: Die Combobox *sorteCB* listet alle Sorten auf

Übung 5.6:

Erstelle in analoger Weise auch für den Anbieter eine Combobox.

(vgl. Projekt *Handel01c*)

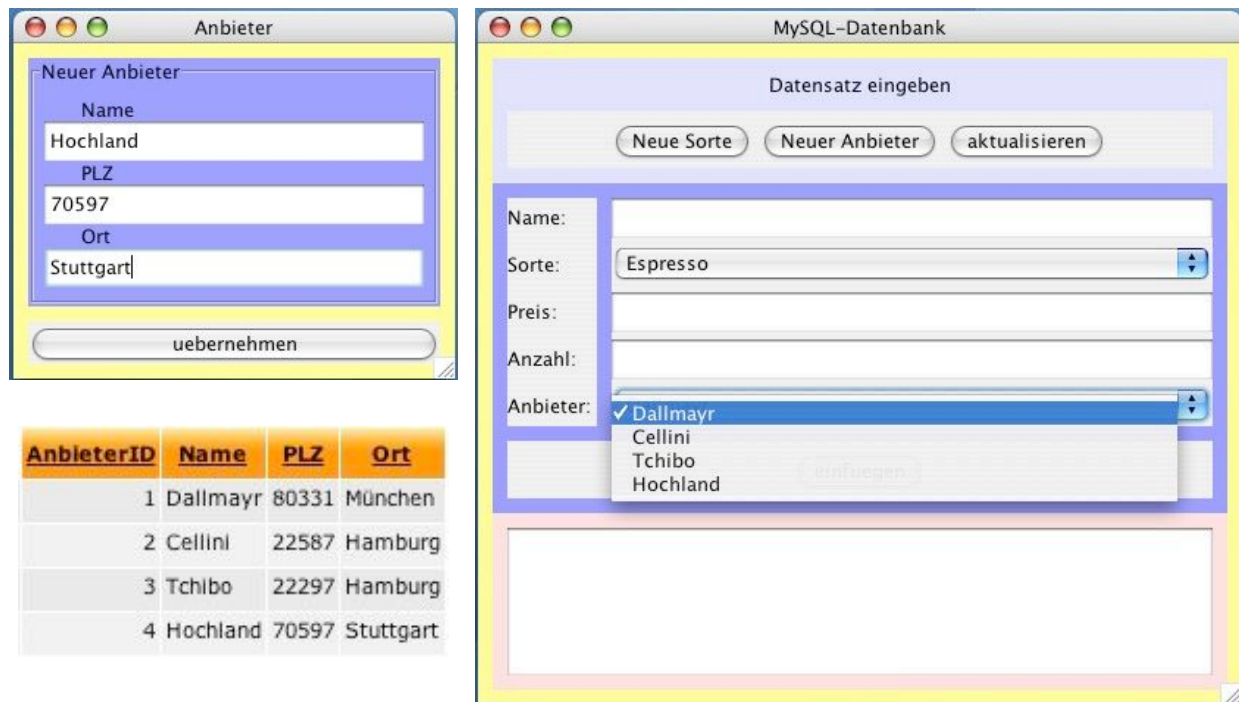


Abbildung 5.13: Die Combobox *anbieterCB* listet alle Anbieter auf

5.1.3 Eingabe eines neuen Kaffees

Nun kannst du auch die Tabelle *Kaffee* mit Datensätzen füllen. Dazu muss die Methode *ein fuegenBActionPerformed()* aus den Textfeldern und den Comboboxen die Werte herauslesen und die SQL-Anweisung *updKaffee* zusammenstellen. Allerdings enthalten die Datenfelder *SorteNr* und *AnbieterNr* der Tabelle *Kaffee* Fremdschlüssel und erwarten somit Zahlen, die den Wert des jeweiligen Primärschlüssels darstellen. Für diese Aufgabe verwendest du die beiden Methoden *holeSorteID()* bzw. *holeAnbieterID()*, die weiter unten beschrieben werden.

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

    String updKaffee = "INSERT INTO Kaffee(Name, SorteNr, Preis, Anzahl,
                                AnbieterNr) " +
        "VALUES (" + anbieter + ", '" + name + "', " +
        "'" + sorte + " ', " + preis + ", " + anzahl + ")";
}
```

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

Abbildung 5.14: Die Methode *ein fuegenBActionPerformed()* in der Klasse *Ansicht*

Wie bereits oben erwähnt, ermittelt die Methode *holeSorteID()* den Wert des Primärschlüssels derjenigen Sorte, die der Benutzer in der Combobox *sorteCB* gewählt hat. Die SELECT-Anweisung *qrySorte* liefert die entsprechende *SorteID* aus der Tabelle *Sorte*. Dieser Wert muss anschließend noch in eine Integer-Zahl konvertiert werden.

```
/** Holt sich die SorteID aus der Tabelle Sorte. */
private int holeSorteID()
{
    String qrySorte = "SELECT SorteID " +
                      "FROM Sorte " +
                      "WHERE Sorte LIKE '" + sorteCB.getSelectedItem() + "'";

    dbAbfrage = new DBAbfrage();
    dbAbfrage.holeDaten(qrySorte);
    Object[][] werte = dbAbfrage.gibWerte();
    //enthalt nur diese eine AnbieterID
    int sorteID = Integer.parseInt("" + werte[0][0]);
    return sorteID;
}
```

Abbildung 5.15: Die Methode *holSorteID()* in der Klasse *Ansicht*

Übung 5.7:

Implementiere die Methoden *ein fuegenBActionPerformed()* und *holeSorteID()* in der Klasse *Ansicht*. Programmiere eine analoge Methode *holeAnbieterID()*.

Wie Abbildung 5.16 zeigt, hat der Benutzer bereits zwei Datensätze in die Tabelle *Kaffee* eingefügt. Die entsprechenden Primärschlüsselwerte der Sorte bzw. des Anbieters werden korrekt in der Tabelle *Kaffee* übernommen.



KaffeeID	Name	SorteNr	Preis	Anzahl	AnbieterNr
1	Espresso dOro	1	9.00	75	1
2	Brasil Mild	2	6.40	50	3

Abbildung 5.16: Der Benutzer hat zwei Datensätze in die Tabelle *Kaffee* eingefügt

5.1.4 Darstellen aller Kaffees in der Tabelle

Zum Schluss sollten alle gespeicherten Kaffees in der Tabelle der GUI dargestellt werden. Die SQL-Anweisung *qryKaffee*

```
"SELECT Kaffee.Name, Anbieter.Name, Sorte.Sorte, Preis, Anzahl " +
"FROM Kaffee, Anbieter, Sorte " +
"WHERE (Anbieter.AnbieterID = Kaffee.AnbieterNr)" +
"AND (Sorte.SorteID = Kaffee.SorteNr)";
```

holt aus allen drei Tabellen die benötigten Informationen. Hieraus kann nun die Tabelle in GUI erstellt werden, wie bereits in Kapitel 3 erklärt.

Übung 5.8:

Implementiere nun die restlichen Methoden, so dass in der GUI die Tabelle alle gespeicherten Kaffees auflistet. Vergleiche hierzu auch die Projekte, die du in Kapitel 4 erstellt hast.

(vgl. Projekt *Handel01d*)

Abbildung 5.17 zeigt nun die erarbeitete Lösung des Projekts *Handel01*.

Name	Sorte	Preis	Anzahl	Name
Espresso dOro	Espresso	7.30	0	Dallmayr
Gran Gusto	Espresso	5.60	25	Cellini
Cafe Picco	Espresso	6.95	0	Hochland
Brasil Mild	Bohnenkaffee	6.40	50	Tchibo
Beste Bohne	Bohnenkaffee	5.30	45	Dallmayr

Abbildung 5.17: Die Tabelle listet alle Kaffees auf

Bemerkung:

In meinem Projekt *Handel01(d)* sind die Methoden *holeSortenamen()* und *holeAnbieternamen()* bzw. die Methoden *holeSorteID()* und *holeAnbieterID()* weitgehend identisch. Hier sollte man sich noch Gedanken darüber machen, wie diese Methoden zusammengefasst werden können

5.2 Implementierung der n:m-Beziehung

Zur Eingabe von Daten in Tabellen, die zueinander in einer 1:m-Beziehung standen, hast du Comboboxen verwendet. Hier konntest du einen Datenwert auswählen. Abbildung 5.1 zeigt, dass die beiden Tabellen Kaffee und Geschmack in einer n:m-Beziehung zueinander stehen. Da nun für einen bestimmten Kaffee mehrere Geschmacksarten gleichzeitig ausgewählt werden können, ist eine solche Combobox nicht mehr anwendbar. Für solche Fälle bieten sich Checkboxen oder Listen an, bei auch mehrere Einträge gleichzeitig ausgewählt werden können. Der Nachteil von Checkboxen ist allerdings, dass deren Anzahl zur Laufzeit nicht verändert werden kann. Wird also eine neue Geschmacksrichtung eingeführt, müsste die grafische Oberfläche neu gestaltet

werden. Deswegen verwende ich für das gleichzeitige Auswählen mehrerer Einträge die Klasse *JList*.

Die Abbildung 5.11 zeigt eine mögliche GUI.

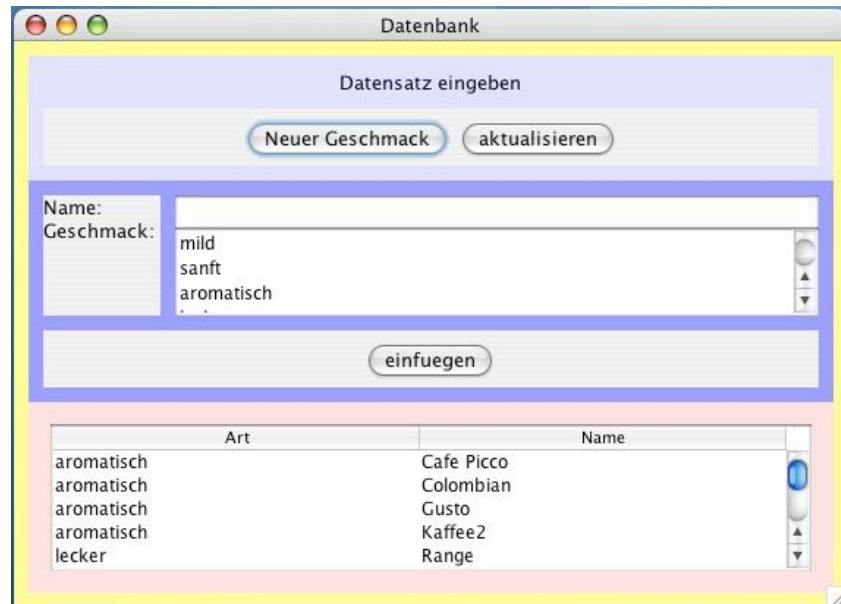


Abbildung 5.18: Eine mögliche grafische Oberfläche

Übung 5.9:

Kopiere in *phpMyAdmin* die Datenbank *Kaffeehandel04* nach *Kaffeehandel05*. Wähle hierbei den Punkt *Nur Struktur*. Somit erhältst du die gleiche Datenbank, deren Tabellen allerdings wieder leer.

Anschließend speichere das Projekt *Handel01* unter dem Namen *Handel02* und stelle den Datenbankzugriff in der Klasse *DBAbfrage* auf die Datenbank *Kaffeehandel5*.

5.2.1 Eingabeformular für neuen Geschmack

Im ersten Schritt erweiterst du die grafischen Benutzeroberfläche um einen neuen Button *Neue Geschmack*. Dieser öffnet ein Eingabeformular, um Tabelle *Geschmack* der Datenbank *Kaffeehandel5* zu füllen.

Übung 5.10:

- a) Implementiere in deinem Projekt *Handel02* eine weitere die Klasse *Geschmack*, die ein weiteres Eingabeformular darstellt.
- b) Implementiere in der Klasse *Ansicht* einen weiteren Button *Neuer Geschmack* und die zugehörige Methode *geschmNeuBActionPerformed()*.

Die grafische Benutzeroberfläche gleicht bereits Abbildung 5.20.

- c) Fülle die Datenbank *Kaffeehandel5* mit Hilfe der drei Eingabeformulare wie in Abbildung 5.19 dargestellt.

SorteID	Sorte
1	Espresso
2	Bohnenkaffee
3	Schonkaffee
4	Pulverkaffee

GeschmackID	Art
1	sanft
2	schonend
3	kräftig
4	aromatisch
5	lecker
6	cremig
7	vollmundig
8	mild

AnbieterID	Name	PLZ	Ort
1	Dallmayr	80331	München
2	Cellini	22587	Hamburg
3	Tchibo	22297	Hamburg
4	Hochland	70597	Stuttgart

Abbildung 5.19: Die vom Benutzer eingegebenen Daten werden in die Tabellen *Sorte*, *Anbieter* und *Geschmack* übernommen

5.2.2 Liste für Geschmack

Abbildung 5.20 zeigt, dass in das *ein fuegenP* eine neue Komponente der Klasse *JList* aufgenommen wird, in der alle Geschmacksrichtungen aufgelistet werden. Wird dieser Liste ein *ListModel* übergeben, so kannst die Einträge ähnlich wie bei den Comboboxen während der Laufzeit ändern. Die Kombination Befehlstaste + Maisklick (Mac-Tastatur) ermöglicht dem Benutzer, verschiedenen Einträge in der Liste zu wählen.

Bemerkung:

Die Anordnung der Labels und der Eingabefelder (Textfelder, Comboboxen und Listen) wird in diesem gewählten Layoutmanager nicht korrekt wiedergegeben. Hier sollte man nach einem besseren Layoutmanager suchen.

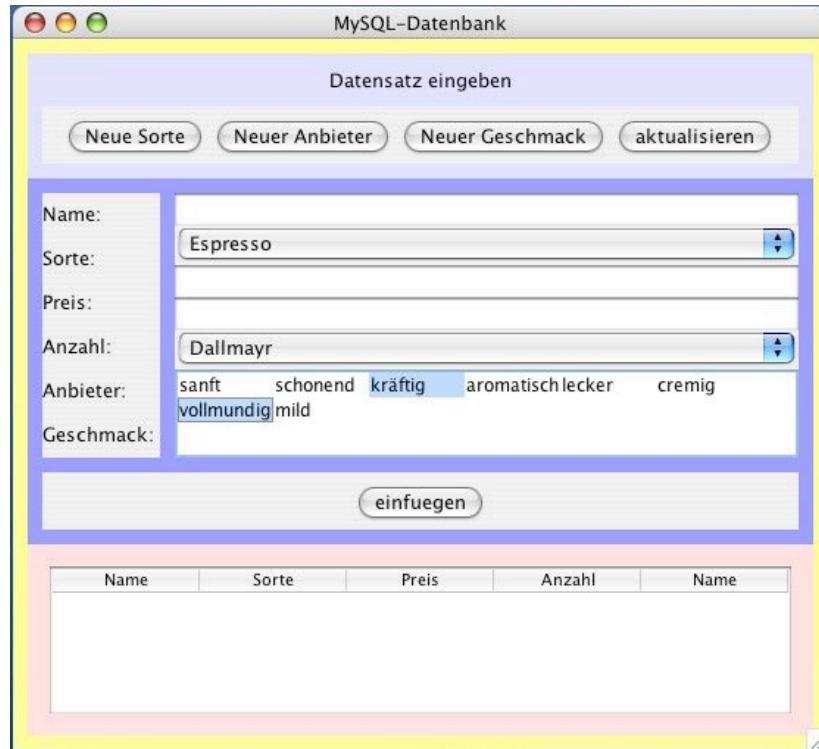


Abbildung 5.20: Die grafische Benutzeroberfläche des Projekts *Handel02*

Du wirst nun lernen, wie

1. eine solche Liste implementiert wird,
2. die Daten aus der Tabelle *Geschmack* in diese Liste übernommen werden.

Zuerst musst du alle neuen Komponenten deklarieren. Dem Katalog der Datenfelder fügst du also das Label *geschmackEinL* und die Liste *geschmackList* hinzu. Für die dynamische Änderung des Listeninhalts wird noch das *geschmackModel* benötigt. Die Liste besitzt keine eigene Funktionalität zum Scrollen der Daten, falls diese nicht vollständig in den Container passen. Deswegen wird die Liste in ein *ScrollPane geschmackScroll* eingebettet, bevor es auf das *einfuegenP* gesetzt wird.

```
private JLabel geschmackEinL;
private JList geschmackList;
private DefaultListModel geschmackModel;
private JScrollPane geschmackScroll;
```

Abbildung 5.21: Zusätzliche Datenfelder in der Klasse *Ansicht*

Bevor die *geschmackList* auf die GUI gesetzt werden kann, muss deren *geschmackModel* mit den Daten aus der Tabelle *Geschmack* gefüllt werden.

Ähnlich wie bei den Comboboxen verwendest du hierfür die Methode *holeGeschmacknamen()*.

```
/** Holt sich alle Geschmacksarten aus der Tabelle Geschmack. */
private void holeGeschmacknamen()
{
    String qryGeschmack = "SELECT Art FROM Geschmack";

    dbAbfrage = new DBAbfrage();
    dbAbfrage.holeDaten(qryGeschmack);
    Object[][] werte = dbAbfrage.gibWerte();

    int anzahl = werte.length;
    for (int i = 0; i < anzahl; i++) {
        geschmackModel.addElement((String) werte[i][0]);
    }
}
```

Abbildung 5.22: Die Methode *holeGeschmacknamen()* in der Klasse *Ansicht*

Nun setzt du innerhalb der Methode *ein fuegenPErstellen()* auf das *ein fuegenP*

1. das Label *geschmackL*
2. die Liste *geschmackList* zusammen mit seinem Modell *geschmackModel*.
Beachte, dass diese Liste in das ScrollPane *geschmackScroll* eingebettet wird.

```
geschmackEinL = new JLabel("Geschmack: ");
labelEinP.add(geschmackEinL);
```

..... weitere Anweisungen

```
geschmackModel = new DefaultListModel();
holeGeschmacknamen();
geschmackList = new JList(geschmackModel);
geschmackList.setSelectionMode(
    ListSelectionMode.MULTIPLE_INTERVAL_SELECTION);
geschmackList.setLayoutOrientation(JList.HORIZONTAL_WRAP);
geschmackList.setVisibleRowCount(-1);
geschmackScroll = new JScrollPane(geschmackList);
geschmackScroll.setPreferredSize(new Dimension(50, 60));
textEinP.add(geschmackScroll);
```

Abbildung 5.23 In der Klasse *Ansicht* werden *geschmackLB* und *geschmackList* auf das *ein fuegenP* gesetzt

Bemerkung:

Hier werden dem *ListModel geschmackList* einige Eigenschaften zugewiesen, die die Benutzerhandhabung beeinflussen:

1. Die Methode *listSelectionModel()* managt die Auswahl der Listeneinträge. Der Benutzer kann in diesem Fall eine beliebige Auswahl der Einträge wählen. Auf der Mac-Tastatur erfolgt diese Auswahl mit Option+Mausklick
2. Die Methode *setLayoutOrientation()* managt die Anordnung der Listeneinträge. Diese werden in diesem Fall horizontal angeordnet mit einem automatischen Zeilenumbruch.
3. Die Methode *setVisibleRowCount()* beeinflusst die Anzahl der sichtbaren Reihen. In diesem Fall wird die maximale Anzahl von Reihen gezeigt, die voreingestellte Größe des Fensters ermöglicht.

Weitere Einzelheiten werden den APIs beschrieben.

Nun hast du bereits beim Öffnen des Fensters alle Werte der Tabellen *Geschmack* in der Liste zur Verfügung und kannst hieraus eine oder mehrere Geschmacksarten auswählen. Im letzten Schritt musst du noch dafür sorgen, dass die Liste immer wieder nach der Eingabe neuer Geschmacksarten wieder aktualisiert wird. Hierzu verwendest du einen dritten Button *aktualisieren* und ergänzt die zugehörige Methode *aktualisierenBtActionPerformed()*.

```
/** Reagiert auf Druecken von aktualisierenB. */  
private void aktualisierenBActionPerformed(ActionEvent e)  
{  
    ..... weitere Anweisungen  
  
    geschmackModel.removeAlLElements();  
    holeGeschmacknamen();  
}
```

Abbildung 5.24: Ergänzung der Methode *aktualisierenBActionPerformed()*

Übung 5.11

Implementiere in deinem Projekt *Handel02* in der Klasse *Ansicht* die Liste *geschmackList* und die Methoden, die in den Abbildungen 5.21 bis 5.24 dargestellt werden.
(vgl. Projekt *Handel02b*)

5.2.3 Eingabe eines neuen Kaffees

Nun kannst du auch die Tabelle *Kaffee* mit Datensätzen füllen.

Wie bereits in Kapitel 5.1.3 besprochen, holt sich die Methode *ein fuegenBActionPerformed()* aus den Textfeldern und den Comboboxen die Werte für den Kaffee, stellt die SQL-Anweisung *updKaffee* zusammen und fügt diesen neuen Kaffee der Tabelle *Kaffee* hinzu.

Im nächsten Schritt musst du nun diesem neuen Kaffee seine Geschmacksarten zuordnen. Da zwischen *Kaffee* und *Geschmack* eine n:m-Beziehung besteht, verwendest du hierzu die Tabelle *Kaff_Geschm*, die nun gefüllt werden muss. In der Methode *ein fuegenBActionPerformed()* holst du zuerst alle in der Liste *geschmackList* ausgewählten Geschmacksarten und sammelst diese in dem Array *selectArt*. Diese Einträge werden in einer Schleife durchlaufen und der Tabelle *Kaff_Geschm* hinzugefügt.

```
Object[] selectArt = geschmackList.getSelectedValues();
```

..... weitere Anweisungen

```
for (int i = 0; i < selectArt.length; i++) {
    String art = (String) selectArt[i];
    ein fuegeKaff_Geschm(name, art);
}
```

Abbildung 5.25: Ergänzung der Methode *ein fuegenBActionPerformed()*

Das Füllen der Tabelle *Kaff_Geschm* erfolgt mit Hilfe der Methode *ein fuegeKaff_Geschm()*. Allerdings enthalten die Datenfelder *KaffeeNr* und *GeschmackNr* der Tabelle *Kaff_Geschm* Fremdschlüssel und erwarten somit Zahlen, die den Wert der jeweiligen Primärschlüssel darstellen. Für diese Aufgabe verwendest du die beiden Methoden *holeKaffeeID()* bzw. *holegeschmackID()*, die weiter unten beschrieben werden. Die INSERT-Anweisung *updKaff_Geschm* kann nun die Tabelle *Kaff_Geschm* füllen.

```
/** Fuegt die aktuellen Daten in die Zuordnungstabelle Kaff_Geschm ein. */
private void ein fuegeKaff_Geschm(String name, String art)
{
    int kaffeeID = holeKaffeeID(name);
    int geschmackID = holeGeschmackID(art);

    String updKaff_Geschm = "INSERT INTO kaff_Geschm(KaffeeNr, GeschmackNr) " +
        "VALUES (" + kaffeeID + ", " + geschmackID + ")";

    dbAbfrage = new DBAbfrage();
    dbAbfrage.neuDaten(updKaff_Geschm);
```

```
}
```

Abbildung 5.26: Die Methode *ein fuegenKaff_Geschm()* in der Klasse *Ansicht*

Wie bereits oben erwähnt, ermittelt die Methode *holeKaffeeID()* den Wert des Primärschlüssels desjenigen Kaffees, den der Benutzer in dem Textfeld *nameEinTF* eingegeben hat und bereits in die Tabelle *Kaffee* eingelesen worden ist. Die SELECT-Anweisung *qryKaffeeID* liefert die entsprechende *KaffeeID* aus der Tabelle *Kaffee*. Dieser Wert muss anschließend noch in eine Integer-Zahl konvertiert werden.

```
/** Holt sich die KaffeeID aus der Tabelle Kaffee. */
private int holeKaffeeID(String name)
{
    String qryKaffeeID = "SELECT KaffeeID " +
                        "FROM Kaffee " +
                        "WHERE Name LIKE '" + name + "'";

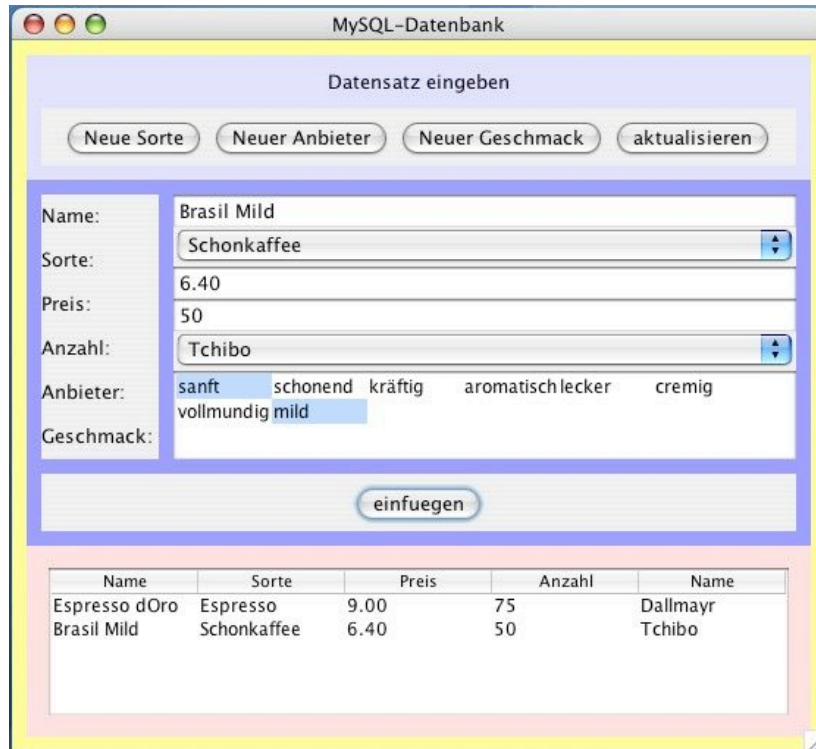
    dbAbfrage = new DBAbfrage();
    dbAbfrage.holeDaten(qryKaffeeID);
    Object[][] werte = dbAbfrage.gibWerte();
    //enthalt nur diese eine GeschmackID
    int kaffeeID = Integer.parseInt("" + werte[0][0]);
    return kaffeeID;
}
```

Abbildung 5.27: Die Methode *holeKaffeeID()* in der Klasse *Ansicht*

Übung 5.12:

Implementiere die Methoden *ein fuegenBActionPerformed()*, *ein fuegenKaff_Geschm()* und *holeKaffeeID()* in der Klasse *Ansicht*.
Programmiere eine analoge Methode *holeGeschmackID()*.

Wie Abbildung 5.28 zeigt, hat der Benutzer bereits zwei Datensätze in die Tabelle *Kaffee* eingefügt. Die entsprechenden Primärschlüsselwerte des Kaffees bzw. des Geschmacks werden korrekt in der Tabelle *Kaff_Geschm* übernommen.



KaffeeID	Name	Sorte
1	Espresso dOro	
2	Brasil Mild	

KaffeeNr	GeschmackNr
2	1
1	3
1	5
2	8

GeschmackID	Art
1	sanft
2	schonend
3	kräftig
4	aromatisch
5	lecker
6	cremig
7	vollmundig
8	mild

Abbildung 5.28: Der Benutzer hat zwei Datensätze in die Tabelle *Kaffee* eingefügt

5.2.4 Darstellen der Geschmacksarten in der Tabelle

Zum Schluss sollten alle gespeicherten Kaffees zusammen mit ihren Geschmacksarten in der Tabelle der GUI dargestellt werden. Die SQL-Anweisung *qryKaffee*

```
"SELECT Kaffee.Name, Sorte.Sorte, Preis, Anzahl, Anbieter.Name,
                                     Geschmack.Art " +
"FROM Kaffee, Sorte, Anbieter, Kaff_Geschm, Geschmack " +
"WHERE (Anbieter.AnbieterID = Kaffee.AnbieterNr) " +
"AND (Sorte.SorteID = Kaffee.SorteNr) " +
```

```
"AND (Kaffee.KaffeeID = Kaff_Geschm.KaffeeNr) " +
"AND (Kaff_Geschm.GeschmackNr = Geschmack.GeschmackID) " +
"ORDER BY Geschmack.Art ASC";
```

holt aus allen fünf Tabellen die benötigten Informationen. Hieraus kann nun die Tabelle in GUI erstellt werden, wie bereits in Kapitel 4 erklärt.

Übung 5.13:

Implementiere nun die restlichen Methoden, so dass in der GUI die Tabelle alle gespeicherten Kaffees auflistet. Vergleiche hierzu auch die Projekte, die du in Kapitel 4 erstellt hast.

(vgl. Projekt *Handel02c*)

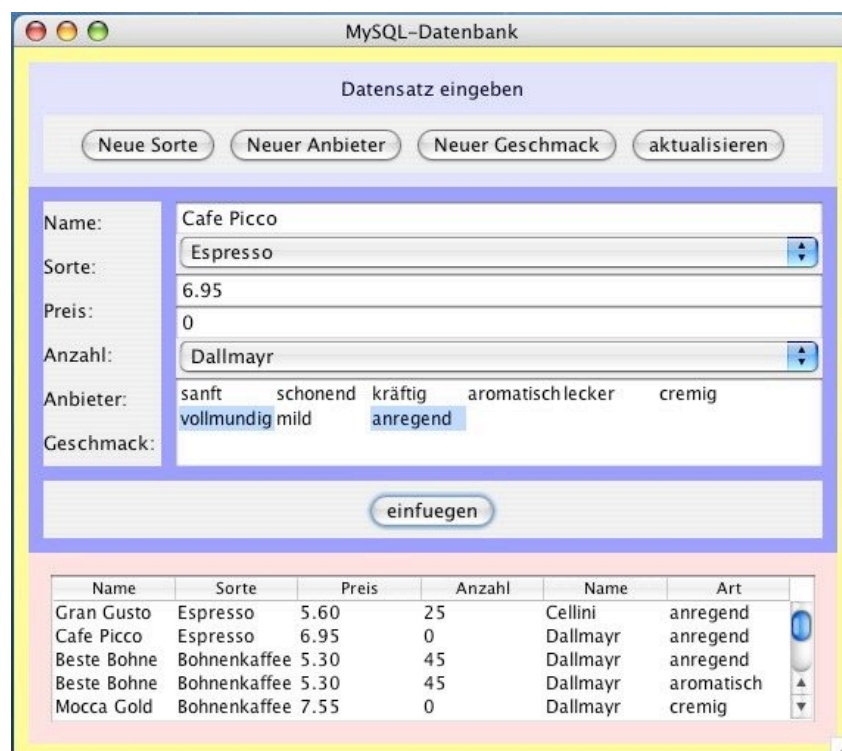


Abbildung 5.29 Die Tabelle listet alle Kaffees auf

5.3 Optimierung des Quelltexts

Bei der Erstellung dieser Klassen tauchte immer wieder die Frage nach der Optimierung des Quelltexts auf. Viele der Methoden, z. B. *holeXxxnamen()*, *holeXxxID()*, usw., besitzen fast den gleichen Quelltext, haben aber trotzdem verschiedene Aufgaben. Auch SQL-Anweisungen werden in verschiedenen Methoden mehrfach implementiert.

Die Frage ist nun, wie weit soll eine Optimierung des Quelltextes bereits bei der Erarbeitung bzw. bei der Einführung eines neuen Lehrstoffs erfolgen. Ich habe mich dafür entschieden, dass es für Schüler sinnvoller ist, die Grundideen des Gesamtkonzepts zu verstehen und deswegen die verschiedenen Aufgaben in mehrere Methoden zu verlagern, auch wenn dann der Quelltext häufig mehrfach programmiert wird.

5.4 Ausblick

5.4.1 Geeignete Layoutmanager

Die Abbildung 5.29 zeigt, dass die Anordnung der Labels und der zugehörigen Textfelder, Comboboxen und Listen nicht mehr korrekt dargestellt wird. Wird jedoch der Layoutmanager *SpringLayout* verwendet, lassen sich die Komponenten wieder korrekt anordnen.

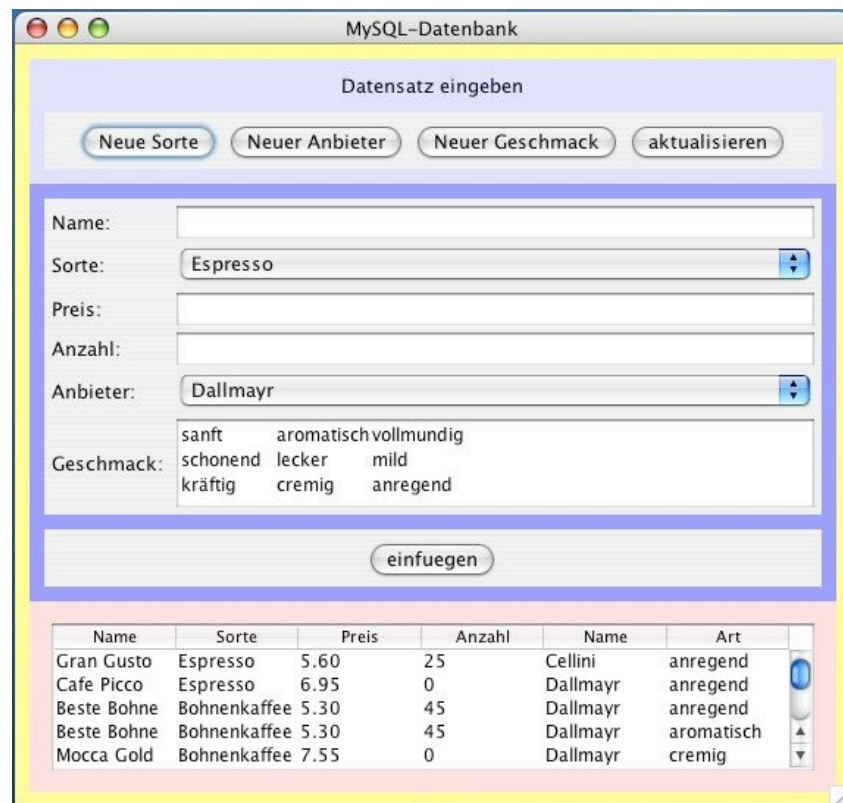


Abbildung 5.30: Projekt *HandelSpring02* verwendet den Layoutmanager *SpringLayout*

In *The JFC Swing Tutorial* wird auf S.501ff die Klasse *SpringUtilities* vorgestellt, die in das Projekt *Handel* mit wenigen Änderungen übernommen werden kann. Abbildung 5.30 zeigt eine mögliche grafische Oberfläche, die den Layoutmanager *SpringLayout* verwendet.

5.4.2 Weitere Formulare

Die Kenntnisse aus diesem Kapitel kannst du nun anwenden, um Formulare zu erstellen, mit deren Hilfe in Datenbank *Kaffeehandel* weitere folgende Aufgaben verrichten:

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

5.4.3 Erzeugen einer Datenbank in Java

Ein weiteres Problem ist, dass bereits eine Datenbank existieren muss. Dem Benutzer sollte die Möglichkeit gegeben werden, auch ohne Verwendung von phpMyAdmin diese Datenbank zu erzeugen.