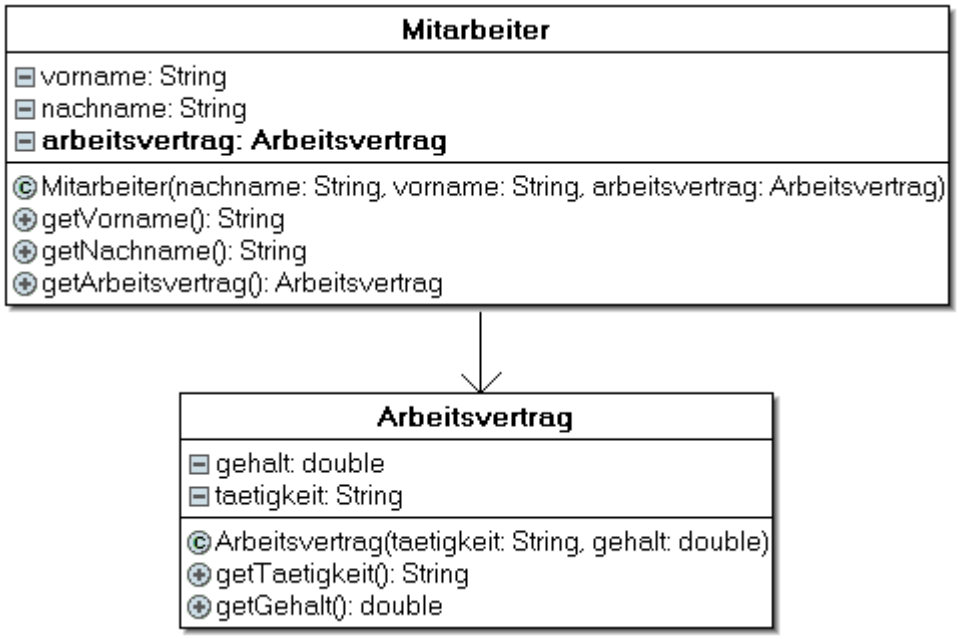
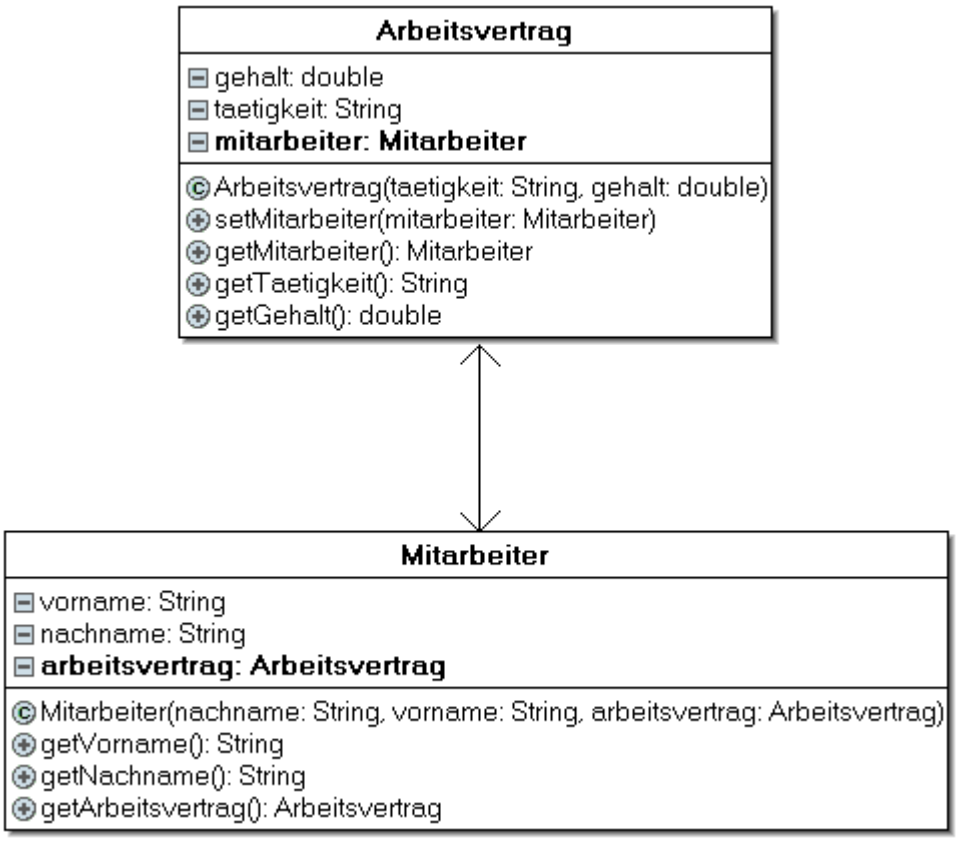
# ***UML - Realisierung von Assoziationen***

*(1.) Das Reisebüro „Titanic Travels“ beauftragt Sie, eine objektorientierte Software zu erstellen, mit welcher u. a. die Daten der Mitarbeiter verwaltet werden.*
*(a.) Die Datenstruktur der Software soll nach folgendem Klassendiagramm erstellt werden:*



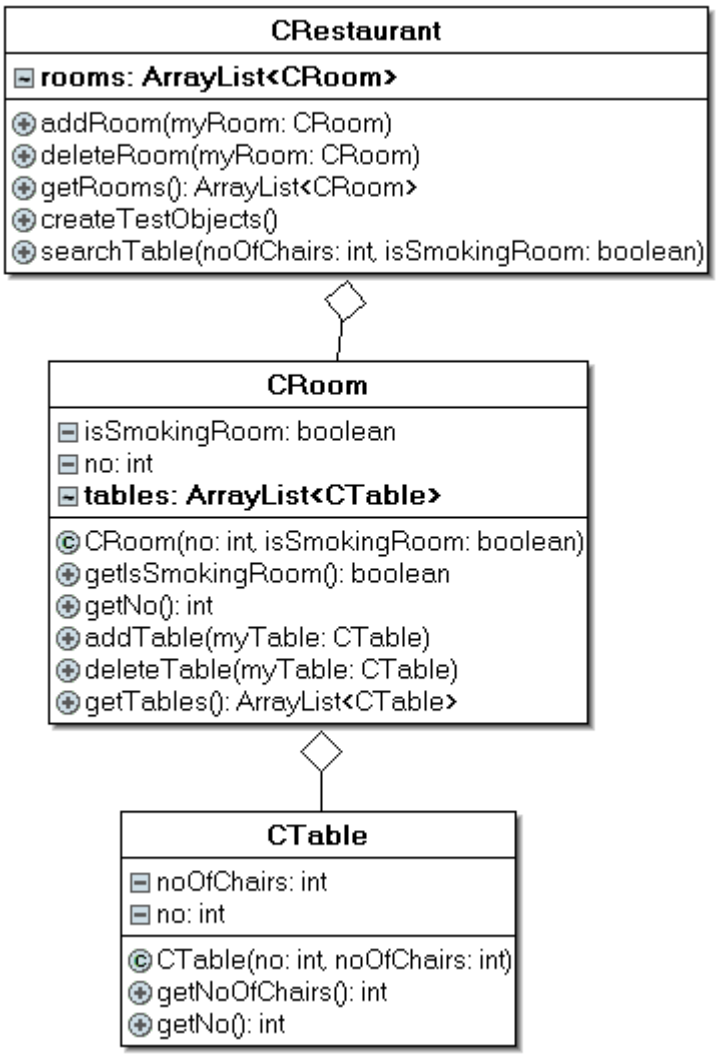

*(b.) Erstellen Sie eine main-Methode, welche genau diejenigen Objekte erzeugt, welche hier im Objektdiagramm dargestellt sind.*

*(c.)Das Klassendiagramm wurde geändert:*




*Ändern Sie den Quellcode so, dass er das Klassendiagramm darstellt!*

*(2.) Es soll eine Software für die Verwaltung (z. B. Tisch- oder Raumreservierungen) des Restaurants „Hüftgold“ erstellt werden. Zu beachten sind folgende Anforderungen: Das Restaurant kann beliebig viele Räume besitzen. In diesen Räumen kann das Rauchen verboten sein oder auch nicht. In jedem Raum gibt es eine beliebige Anzahl an Tischen. Alle Tische haben eine beliebige Anzahl an Stühlen. Für die Datenstruktur der Software wurde bereits ein Klassendiagramm erstellt:*




*(a.) Setzen Sie das Klassendiagramm in den Javacode um, berücksichtigen Sie dabei folgende Festlegungen: Alle Klassen besitzen einen Konstruktor, welcher alle Attribute setzt. Für jedes Attribut (alle Klassen) existiert eine get-Methode.*
*Für die Umsetzung der 1:n - Beziehungen ist die JDK-Klasse ArrayList zu verwenden. Die entsprechenden Referenzattribute (Listenobjekte CRoom bzw. CTable) sind bereits im Klassendiagramm enthalten. Zur Verwaltung der Referenzattribute (Listen) sind jeweils drei Methoden im Klassendiagramm definiert, welche folgende Operationen ermöglichen:*

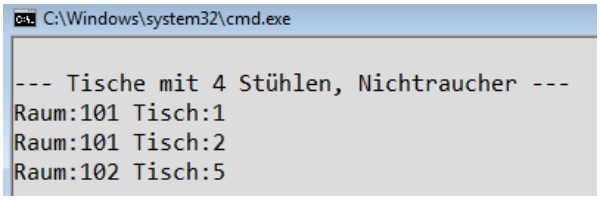
- *Objekt zur Liste hinzufügen*
- *Objekt in der Liste löschen*
- *komplette Liste zurückgeben*

*Die Methode createTestObjects soll 3 Räume mit 9 Tischen erzeugen!*
*Die Methode searchTable wird in (c.) erstellt!*
*(b.) Erstellen Sie ein Objektdiagramm für folgende Situation:*
*Das Restaurant besitzt 2 Räume. Im Raum 1 stehen 2 Tische mit je 6 Stühlen, hier darf nicht geraucht werden. Im Raum 2 steht 1 Tisch mit 8 Stühlen, das Rauchen ist hier erlaubt. Für geschlossene Gesellschaften ist nur der Raum 2 geeignet.*

*(c.) Implementieren Sie in der Klasse Restaurant diese Methode:*

*public void searchTable(int noOfChairs, boolean isSmokingRoom)*

*Die Methode dient zur Suche nach Tischen, welche bestimmte Kriterien erfüllen. Diese Kriterien (Anzahl der Stühle, Rauchen ja/nein) werden der Methode beim Aufruf übergeben. Die Ergebnisse sind auf der Konsole auszugeben, Beispiel:*



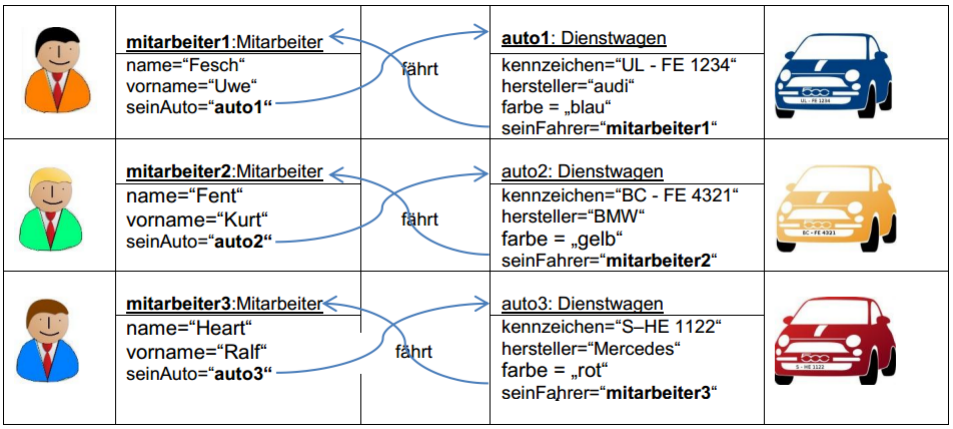
*(3.) Ein Händler hat viele Fahrzeuge in seinem Sortiment, welche alle einen Namen und einen Preis besitzen. Sämtliche Fahrzeuge gehören zu jeweils einer von zwei Fahrzeuggruppen, es sind entweder PKW und LKW.*
*Für jeden PKW muss die Anzahl der Türen gespeichert werden. Die Größe der Ladefläche (in m²) ist ein Merkmal, was für jeden LKW gespeichert werden muss.*
*(a.) Erstellen Sie ein entsprechendes Klassendiagramm. Innerhalb der Klassen müssen keine Konstruktoren bzw. Methoden angegeben werden. Nutzen Sie sinnvolle Vererbungsstrukturen.*
*(b.) Überführen Sie das in (a.) erstellte Klassendiagramm in den Javacode, beachten Sie dabei folgende Hinweise:*
*Ihr Code muss mit der abgebildeten Testklasse funktionieren. Die Klassen CCar und CTruck müssen eine Methode toString besitzen. Diese gibt alle gespeicherten Attributwerte in Form eines einzigen Strings zurück.*
*Die jeweiligen Mengen an PKW und LKW, welche im Listenobjekt gespeichert sind, sollen ausgegeben werden.*
*Hinweis: Um festzustellen, ob ein Objekt zu einer bestimmten Klasse gehört, kann der Operator instanceof verwendet werden!*

Konsolenausgabe:
```
In der Liste befinden sich
2 PKW und 1 LKW.
```

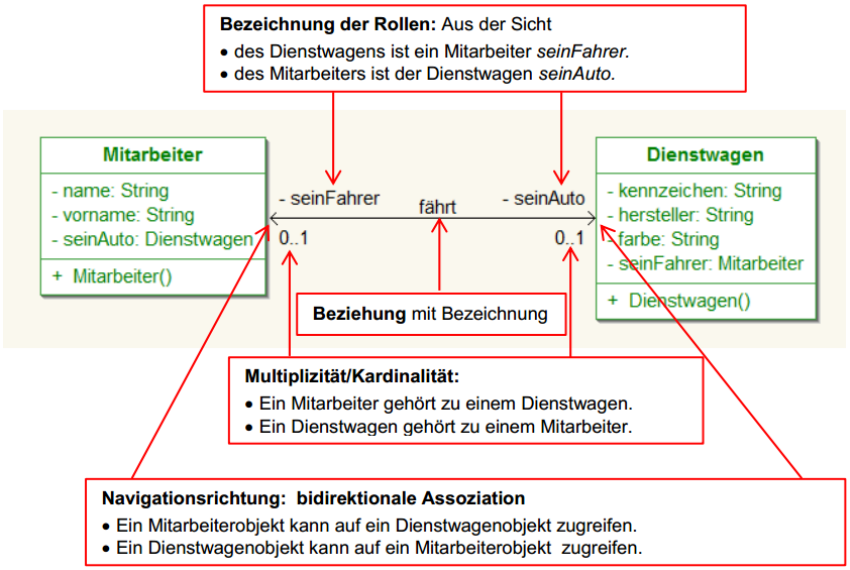
```
import java.util.ArrayList;

public class CTestCarsAndVehicles
{
  public static void main (String args[])
  {
    //CDealer-Objekt anlegen
    CDealer myDealer = new CDealer();
    // Create Car and add it to the list
    // Bedeutung der Übergabewerte: Name, Preis, Anzahl der Türen
    CCar myCar = new CCar("Mazda6", 30000.00, 4);
    myDealer.addVehicle(myCar);
    // Create Truck and add it to the list
    //Bedeutung der Übergabewerte: Name, Preis, Ladeflaeche in m²
    CTruck myTruck = new CTruck("IFA W50", 99500.00, 7.5);
    myDealer.addVehicle(myTruck);
    // Create 2nd Car and add it to the list
    // Bedeutung der Übergabewerte: Name, Preis, Anzahl der Türen
    myCar = new CCar("Trabant 601", 55777.00, 2);
    myDealer.addVehicle(myCar);

    // Output of all our vehicles
    // Get list of vehicles
    ArrayList<CVehicle> liste = myDealer.getVehiclelist();
    //gespeicherte CVehiclee (Attributwerte) ausgegeben:
    for(int i = 0; i < liste.size(); i++)
      System.out.println(liste.get(i).toString());

    // Counts the no of Cars & Trucks
    int noOfCars = 0, noOfTrucks= 0;

    for(int i = 0; i < liste.size(); i++)
      if(liste.get(i) instanceof CCar)
        noOfCars++;
      else
          noOfTrucks++;
    System.out.println("\nWe have:");
    System.out.println(noOfCars + " Cars & " + noOfTrucks + " Trucks!");
  }
}
```

*(c.) Die Fahrzeuge werden nun in einer verketteten Liste gespeichert. Ändern Sie die Klasse Fahrzeug, so, dass ihre eigene erstellte Klasse verwendet wird und zeichnen Sie ein Klassendiagramm!*

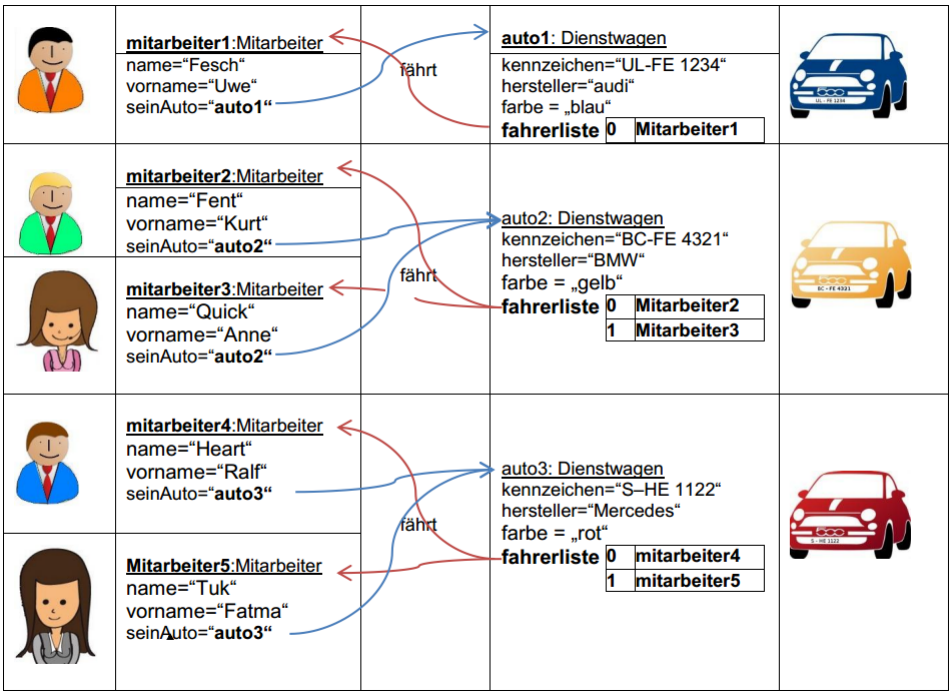
*(4.) In einer Firma „SurelockHolmes“ Außendienstmitarbeitern Geschäftswagen zur Verfügung. Folgende drei Szenarien sind denkbar, die als Klassen und einer Testklasse kodiert werden sollen! Die Grundstruktur der Klassen befindet sich im Ordner 04\Vorlage!*
*(a.) Jeder Mitarbeiter/in fährt immer mit demselben Auto, ein Auto wird auch immer vom selben Mitarbeiter gefahren Von den Mitarbeiterobjekten aus kann auf Autoobjekte zugegriffen werden und umgekehrt:*



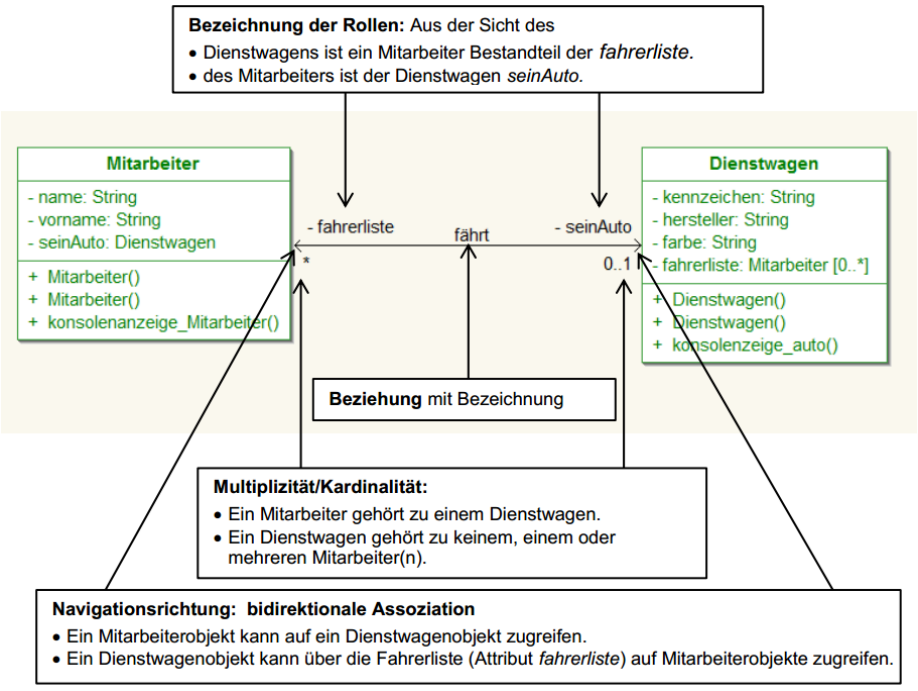

- *Soll von den Mitarbeiterobjekten aus auf die Autoobjekte zugegriffen werden, geschieht dies über das Referenzattribut seinAuto, dessen Attributwert das referenzierte Autoobjekt ist.*
- *Soll von den Autoobjekten aus auf die Mitarbeiterobjekte zugegriffen werden, geschieht dies über das Referenzattribut seinFahrer, dessen Attributwert der referenzierte Mitarbeiter ist.*



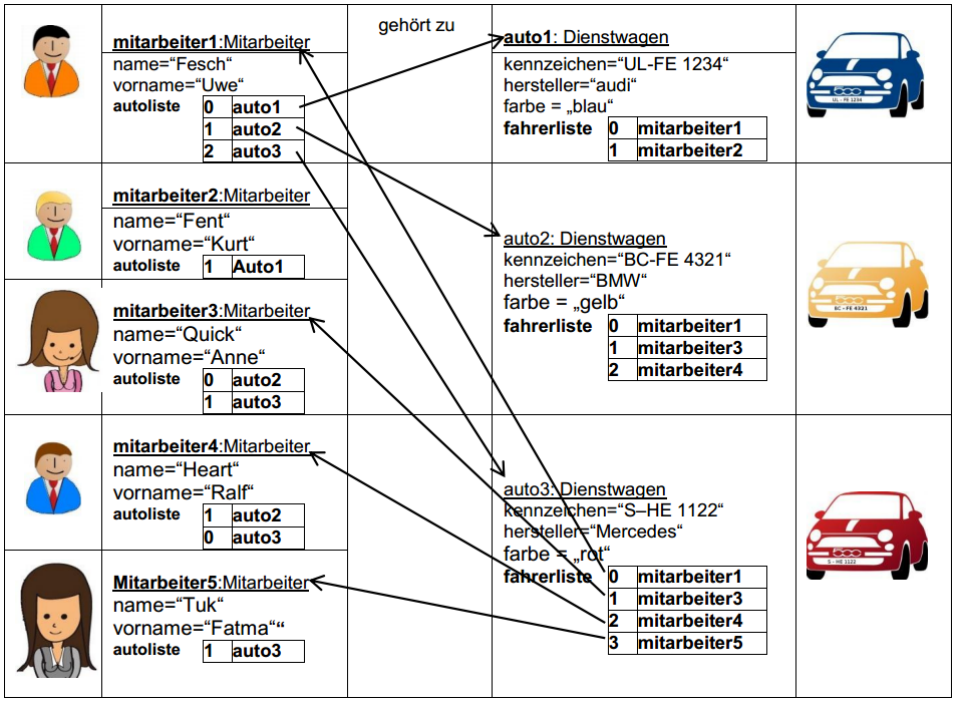
*(b.) Bisher konnte jeder Mitarbeiter immer nur ein und dasselbe Auto benutzen. Da sich in den Absatzgebieten Oberschwaben/Bodensee und Stuttgart die Nachfrage stark erhöht hat, werden für diese Absatzgebiete jeweils eine Mitarbeiterin (Anne Quick für Oberschwaben/Bodensee und Fatma Tuk für Stuttgart) eingestellt. Da immer nur ein Mitarbeiter/eine Mitarbeiterin in diesen beiden Absatzgebieten Außendienst macht und der/die andere dann im Innendienst arbeitet, teilen sich Kurt Fend und Anne Quick sowie Ralf Heart und Fatma Tuk jeweils einen Dienstwagen:*



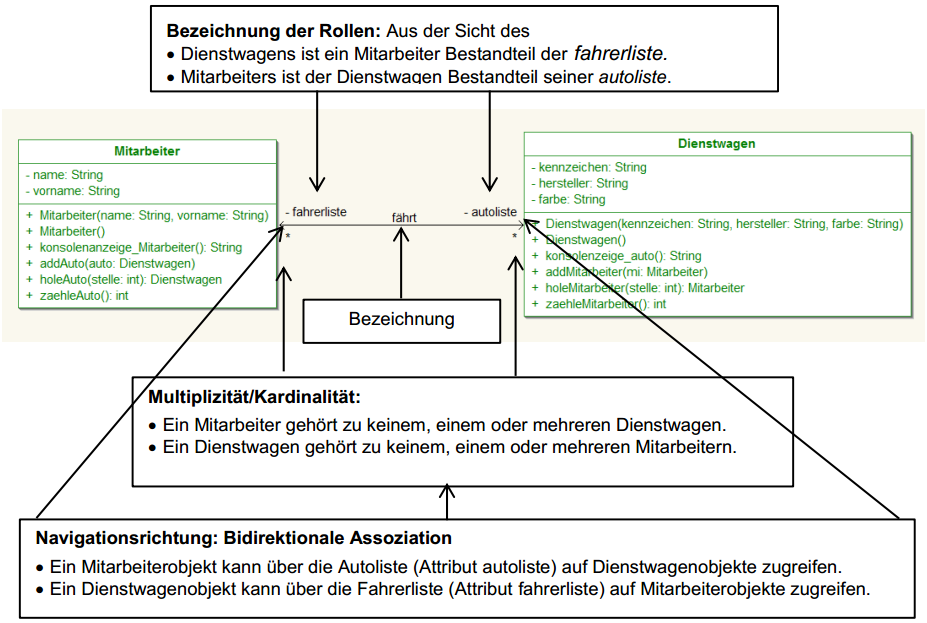

- *Soll von einem Mitarbeiterobjekt aus auf ein Autoobjekt zugegriffen werden, geschieht dies über das Referenzattribut seinAuto, dessen Attributwert das referenzierte Autoobjekt ist.*
- *Sollen von einem Autoobjekt aus auf ein oder mehrere Mitarbeiterobjekte zugegriffen werden, geschieht dies über die Referenzliste fahrerliste, in der als Werte die referenzierten Mitarbeiterobjekte enthalten sind*



*(c.) Bisher stand den Mitarbeitern in jedem Absatzgebiet jeweils ein Dienstwagen zur Verfügung. Aus organisatorischen Gründen entschloss sich die Geschäftsleitung, den Getränkeabsatz im Standort Ehingen zu zentralisieren.*
*Jedem Mitarbeiter steht jetzt jeder Dienstwagen zur Verfügung. Es soll möglich sein, zu ermitteln, welcher Mitarbeiter mit welchem Dienstwagen gefahren ist, ebenso sollen nach wie vor für jeden Dienstwagen die Mitarbeiter aufgelistet werden können, die mit ihm gefahren sind.*
*Die untenstehende Darstellung zeigt, welche Dienstwagen der Mitarbeiter Uwe Fesch (Objekt mitarbeiter1) im Monat Mai 20XX gefahren hat und welche Mitarbeiter den Dienstwagen3 (Objekt auto3) mit dem Kennzeichen S – HE 1122 benutzt haben*




- *Soll von einem Mitarbeiterobjekt aus auf ein Autoobjekt zugegriffen werden, geschieht dies über das Referenzattribut seinAuto, dessen Attributwert das referenzierte Autoobjekt ist Soll von einem Mitarbeiterobjekt aus auf ein oder mehrere Dienstwagenobjekte zugegriffen werden, geschieht dies über die Referenzliste autoliste, in der als Werte die referenzierten Dienstwagenobjekte enthalten sind.*
- *Soll von einem Autoobjekt aus auf einoder mehrere Mitarbeiterobjekte zugegriffen werden, geschieht dies über die Referenzliste fahrerliste, in der als Werte die referenzierten Mitarbeiterobjekte enthalten sind.*



*(d.) Um einen besseren Überblick über die Benutzung der Dienstfahrzeuge zu erhalten, wünscht die Vertriebsleitung, dass neben der Nutzung der Fahrzeuge zusätzlich noch festgehalten wird, an welchem Termin welcher Mitarbeiter ein Auto fuhr und wie viele Kilometer zurückgelegt wurden. Außerdem soll festgehalten werden, ob die Fahrt dienstliche Gründe hatte oder nicht.*
*Das Fahrtenbuch für den Dienstwagen 3 (Objekt auto3) weist beispielsweise folgende Eintragungen auf:*

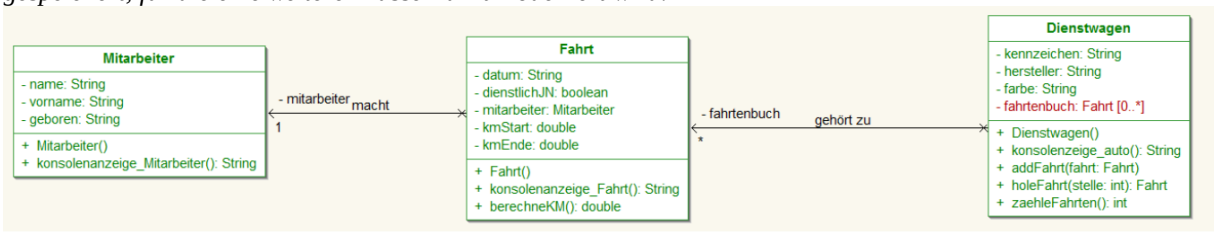
GeLa GmbH Fahrtenbuch Seite 1

| Dienstwagen: | S – HE 1122 | Fabrikat | Mercedes | Farbe | rot |
|---|---|---|---|---|---|

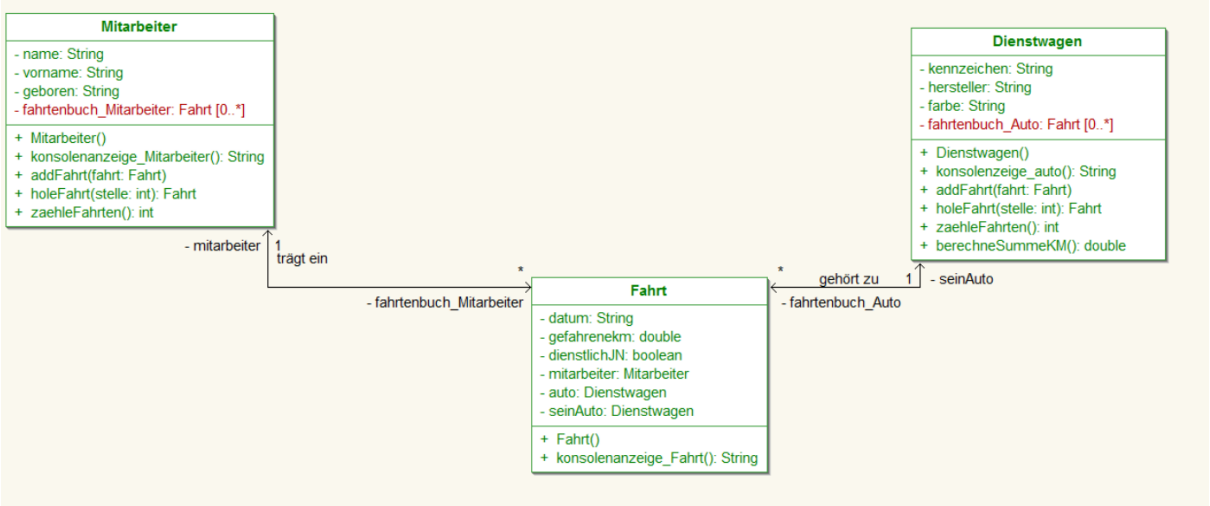
**Fahrten:**

| Nr | Datum | Name, Vorname | KM Start | KM Ende | dienstlich (X) |
|---|---|---|---|---|---|
| 1 | 18.04.2013 | Heart, Ralf | 10500 | 11000 | |
| 2 | 21.04.2013 | Quick, Anne | 11000 | 11300 | |
| 3 | 22.04.2013 | Fesch, Uwe | 11300 | 11450 | |
| 4 | 23.04.2013 | Tuk, Fatma | 11450 | 12050 | X |

*Würden die Attribute datum, gefahrenekm und dienstlich den Mitarbeitern oder den Dienstwagen zugeordnet, könnten für jedes Mitarbeiterobjekt beziehungsweise für jedes Dienstwagenobjekt nur Werte für eine Fahrt gespeichert werden. Die nicht eindeutig zuordenbaren Attribute werden deshalb in eigenen Objekten gespeichert, für die eine weitere Klasse Fahrt modelliert wird.*

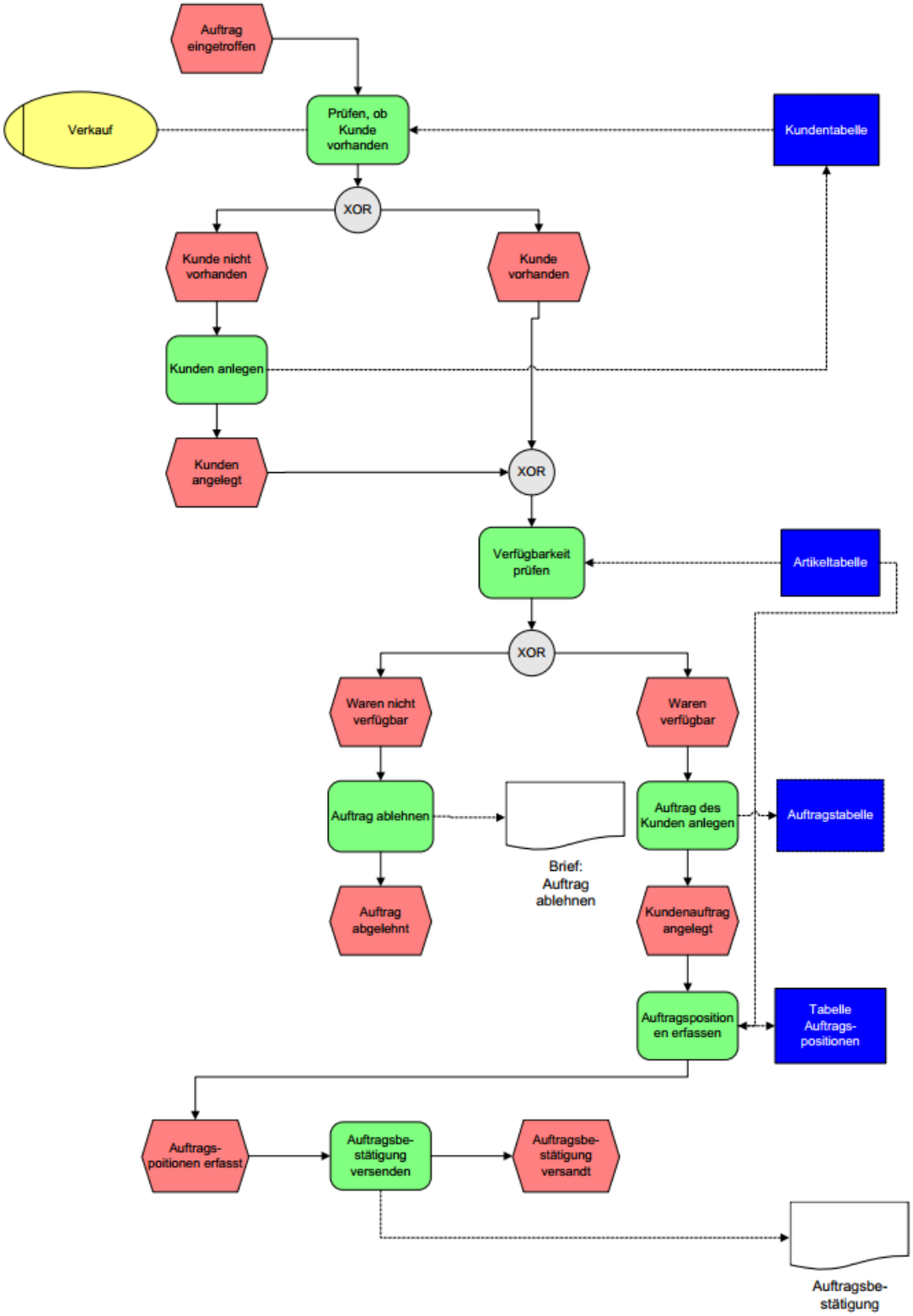



*Erläuterung:*
*In der Klasse Dienstwagen werden die Fahrten im Attribut fahrtenbuch gesammelt. Das Attribut fahrtenbuch ist ein Objekt der Klasse ArrayList. Dadurch können mehrere Fahrten gespeichert werden. Es besteht zur Klasse Fahrt eine unidirektionale Assoziation mit der Multiplizität N. Die Klasse Fahrt enthält zur Klasse Mitarbeiter das Referenzattribut mitarbeiter vom Typ der Klasse Mitarbeiter. Es besteht also von der Klasse Fahrt zur Klasse Mitarbeiter eine unidirektionale Assoziation mit der Multiplizität 1.*
*Im Attribut fahrtenbuch werden die Objekte der Klasse Fahrt aufgenommen. Auf die Daten der Mitarbeiter kann dann über das Attribut mitarbeiter der Klasse Fahrt zugegriffen werden.*
*Die Klasse Fahrt verfügt sowohl über die Eigenschaften einer Klasse als auch über die einer Assoziation, da sie sowohl mit der Klasse Mitarbeiter als auch mit der Klasse Dienstwagen verbunden ist. Man spricht daher auch von einer attribuierten Assoziation beziehungsweise von einer Assoziationsklasse*
*Im oben dargestellten Klassendiagramm besteht jeweils eine unidirektionale Assoziation von der Klasse Dienstwagen zur Klasse Fahrt und von der Klasse Fahrt zur Klasse Mitarbeiter.*
*- Mit einem Auto (Objekt der Klasse Dienstwagen) werden keine, eine oder mehrere Fahrten gemacht.*
*- Ein Eintrag im Fahrtenbuch (Objekt der Klasse Fahrt) wird immer von einem Mitarbeiter gemacht.*
*Von den Mitarbeiterobjekten aus kann nicht auf Objekte der Klasse Fahrt und von den Objekten der Klasse Fahrt nicht auf Objekte der Klasse Dienstwagen zugegriffen werden.*
*Denkbar wären auch bidirektionale Assoziationen zwischen den Klassen Mitarbeiter und Fahrt und den Klassen Dienstwagen und Fahrt. In diesem Falle müssen auch die Mitarbeiter ein Fahrtenbuch führen (Attribut fahrtenbuch_Mitarbeiter vom Typ ArrayList). Außerdem muss in der Klasse Fahrt das Attribut auto vom Typ Dienstwagen eingefügt werden (siehe nachfolgendes UML-Klassendiagramm).*



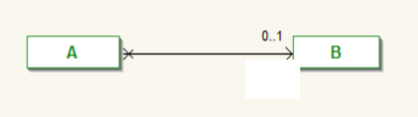
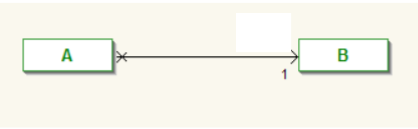
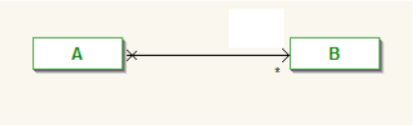
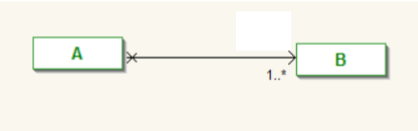

*Im erweiterten Modell*
*- kann ein Mitarbeiter (Objekt der Klasse Mitarbeiter) keine, eine oder mehrere Fahrten machen.*
*- gehört ein Eintrag im Fahrtenbuch des Mitarbeiters (Objekt der Klasse Fahrt) immer zu einem Auto.*
*Kodieren Sie das Modell mit den unidirektionalen Verbindungen!*

*(5.)*

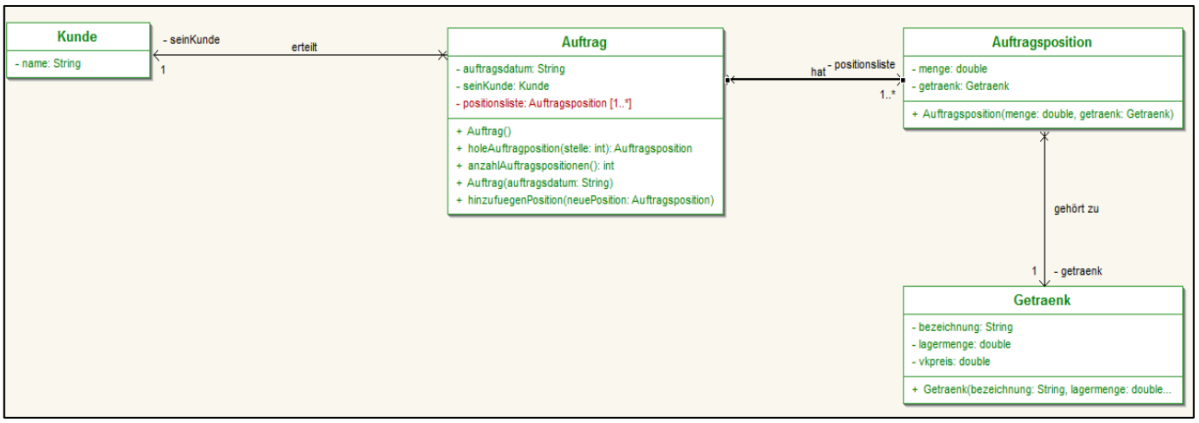



*Wenn der Kundenauftrag eingetroffen ist, wird in der Verkaufsabteilung geprüft, ob die Kundendaten vorhanden sind. Wenn sie nicht vorhanden sind, werden sie neu angelegt. Wenn die Kundendaten angelegt sind oder schon vorhanden waren, kann die Verfügbarkeit der Artikel geprüft werden. Wenn nicht genügend Waren verfügbar sind, muss der Auftrag abgelehnt und dem Kunden die Ablehnung mitgeteilt werden. Sind die Waren verfügbar, kann der Auftrag angelegt und die Auftragspositionen können erfasst werden. Jede Auftragsposition bezieht sich auf einen Artikel. Wenn die Auftragspositionen erfasst sind, wird eine Auftragsbestätigung erzeugt und an den Kunden versandt.*

*Übersicht über Kann- und Muss-Assoziationen:*

| | Beschreibung |
|---|---|
|  | **Kann-Assoziation mit Muliplizität 1**<br><br>Zu einem Objekt der Klasse *A* gibt es kein oder ein assoziiertes Objekt der Klasse *B*. |
|  | **Muss-Assoziation mit Muliplizität 1**<br><br>Zu einem Objekt der Klasse *A* gibt es ein assoziiertes Objekt der Klasse *B*. |
|  | **Kann-Assoziation mit Muliplizität N**<br><br>Zu einem Objekt der Klasse *A* gibt es kein, ein oder mehrere assoziierte Objekte der Klasse *B*. |
|  | **Muss-Assoziation mit Muliplizität N**<br><br>Zu einem Objekt der Klasse *A* gibt es ein oder mehrere assoziierte Objekte der Klasse *B*. |

*Aus dem Geschäftsprozess (siehe Seite 59) wurde das folgende (vereinfachte) Klassendiagramm entworfen. Das Modell ist aus didaktisch-methodischen Gründen reduziert. Nur die Klassen und deren Attribute, die zum Verständnis unerlässlich sind, werden verwendet:*




*Die Klasse Auftrag steht im Mittelpunkt. Wie in der ereignisgesteuerten Prozesskette beschrieben, kann ein Auftrag erst angelegt werden, wenn die Kundendaten vorliegen und er mindestens eine Position hat. Eine Auftragsposition muss immer zu einem Artikel führen.*
*Aus diesem Sachverhalt ergeben sich folgende Assoziationen:*
*- Von der Klasse Auftrag zur Klasse Kunde besteht somit eine unidirektionale MussAssoziation mit der Multiplizität 1. Die Assoziation wird mit dem Referenzattribut seinKunde realisiert.*
*- Von der Klasse Auftrag zur Klasse Auftragsposition besteht eine unidirektionale MussAssoziation mit der Multiplizität N und wird mit dem Referenzattribut positionsliste realisiert.*
*- Von der Klasse Auftragsposition zur Klasse Getraenk besteht eine Muss-Assoziation mit der Multiplizität 1. Sie wird mit dem Referenzattribut getraenk realisiert. Bei einer Muss-Assoziation ist Multiplizität größer als Null. Dies bedeutet, dass mindestens ein Objekt assoziiert sein muss. Eine Muss-Assoziation kann programmiertechnisch umgesetzt werden, indem*
*- der Konstruktor des Objekts ein bereits existierendes Objekt der assoziierten Klasse als Übergabewert einfordert oder*
*- mit dem Konstruktor des Objektes das Referenzattribut in der assoziierten Klasse erzeugt wird.*

*Problemstellung:*
*In der Auftragsbearbeitung der GeLa GmbH geht ein Auftrag des Kunden Finke KG zur Lieferung von 30 Flaschen Apfelschorle, 20 Flaschen Mineralwasser Blauquelle natur und 10 Flaschen Himbeersaft ein.*
*Nachdem geprüft ist, dass der Kunde schon angelegt wurde und die Getränke verfügbar sind soll der Auftrag mithilfe einer objektorientierten Software angelegt sowie die Auftragsbestätigung ausgedruckt und versandt werden.*
*Folgende Aufgabenstellungen sind zu erledigen:*
*1. Ein Javaprojekt projektGeLa_Auftragsbearbeitung und darin ein Paket paketGela_Auftragsbearbeitung sind anzulegen und das UML-Klassendiagramm mit den Klassen, Attributen und Assoziationen beschrieben, ist zu erstellen.*
*2. Eine Startklasse mit der main-Methode ist einzurichten und als Daten das Kundenobjekt kunde1 „Finke Getränke KG“ sowie die folgenden Getränkeobjekte getraenk1 bis getraenk5 anzulegen. Zum Anlegen der Getränkeobjekte ist ein Konstruktor zu verwenden, der die Daten des jeweiligen Getränks als Übergabewert fordert:*

| *Bezeichnung* | *Menge* | *Verkaufspreis* | |
|---|---|---|---|
| *Cola light,* | *0,3l* | *500* | *1.10* |
| *Lemonlimonade,* | *0,3l* | *500* | *0.90* |
| *Apfelschorle,* | *0,5l* | *1000* | *1.00* |
| *Blauquelle nat.,* | *0,7l* | *1000* | *0.45* |
| *Himbeersaft,* | *1,0l* | *300* | *1.75* |

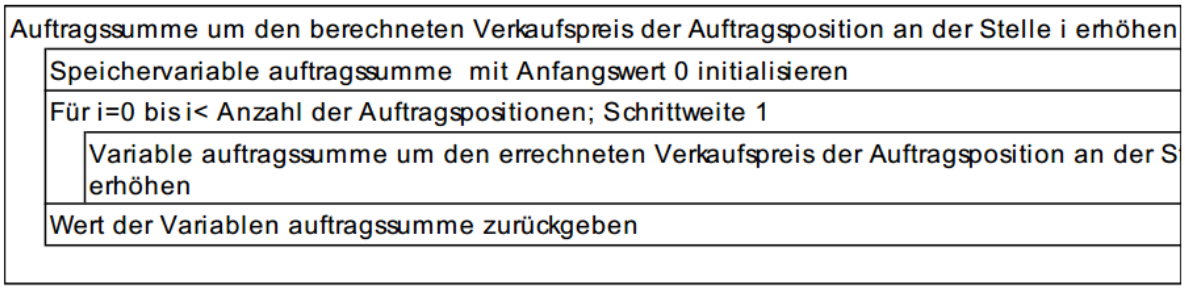
*3. Anpassen des Konstruktors der Klasse Auftrag und Anlegen eines Auftragsobjekts in der Startklasse.*
*Wie im Modell auf Seite 61 beschrieben, bezieht sich ein Auftrag auf einen Kunden und er wird nur dann angelegt, wenn er mindestens eine Auftragsposition hat.*
*3.1. Der Konstruktor der Klasse Auftrag ist so anzupassen, dass er neben dem Auftragsdatum*
*- das bereits existierendes Kundenobjekt kunde1 der assoziierten Klasse Kunde als Übergabewert einfordert und*
*- ein Objekt positionsliste der Klasse ArrayList erzeugt wird, in das die Objekte der assoziierten Klasse Auftragsposition aufgenommen werden können.*
*3.2. Anschließend ist der Auftrag des Kunden Finke Getränke KG als Objekt auftrag1 in der Startklasse anzulegen.*
*4. Die Objekte pos1, pos2 und pos3 der Klasse Auftragsposition sind anzulegen und anschließend in das Attribut positionsliste aufzunehmen. Wie im Modell auf Seite 61 beschrieben ist, muss sich eine Auftragsposition auf ein Getränk beziehen.*
*4.1. Der Konstruktor der Klasse Auftragsposition muss angepasst werden. Er muss neben der verkauften Menge das im Auftrag angegebene Objekt aus der Klasse Getraenk verlangen.*
*4.2. Anschließend werden in der Startklasse die Objekte der Klasse Auftragsposition erzeugt und in das Attribut positionsliste hinzugefügt.*
*5. Nach dem die Auftragsdaten erfasst sind, ist die nachfolgend dargestellte Auftragsbestätigung im Konsolenfenster anzuzeigen.*

```
Auftragsbestätigung

Auftrag des Kunden Finke Getränke KG vom 31.07.2013

Artikel:                Menge   VKP/St. Gesamt
-------------------------------------------------
Apfelschorle, 0,5l      30.0    1.0     30.0
Blauquelle nat., 0,7l   20.0    0.45    9.0
Himbeersaft, 1,0l       10.0    1.75    17.5
-------------------------------------------------
Auftragssumme:                          56.5
```

Auftragskopf

Auftragspositionen

*Dazu müssen folgende Aufgabenstellungen bearbeitet werden:*
*5.1. In der Klasse Auftragsposition ist die Methode public double berechneVerkaufspreis() zu erstellen. Die Methode liefert als Ergebnis das Produkt aus der verkauften Menge (Attribut menge) und dem Verkaufspreis (Attribut vkpreis des assoziierten Objekts der Klasse Getraenk).*
*5.2. In der Klasse Auftrag ist die Methode public double berechneAuftragssumme() zu erstellen. Die Methode liefert als Ergebnis die Summe der errechneten Verkaufspreise aller Positionen des Auftrages. Dazu wird der berechnete Gesamtverkaufspreis jeder Position ermittelt und aufsummiert.*
*Struktogramm der Methode public double berechneAuftragssumme()*




*5.3. Die Auftragsbestätigung (siehe Abb. oben) ist im Konsolenfenster darzustellen. Die Konsolenanzeige soll so aufgebaut werden, dass die Daten der Auftragsbestätigung jeweils als Ausgabestring von Methoden aus den Fachklassen geliefert werden.*
*5.3.1. Die Methode String konsolenanzeigeAuftragsposition() der Klasse Auftragsposition liefert die Daten einer Auftragsposition.*
*5.3.2. Die Methode String konsolenanzeigeAKop() der Klasse Auftrag liefert die Daten des Auftragkopfes.*
*5.3.3. Die Methode String konsolenanzeigeAPos() der Klasse Auftrag liefert die Daten aller Auftragspositionen eines Auftrags. Die beschriebene Vorgehensweise vereinfacht den Inhalt der Startklasse, insbesondere, wenn mehrere Aufträge erfasst werden sollen.*

*(6.) Gegeben ist nachfolgender Quellkode! Erstellen Sie dazu das zugehörige Klasssendiagramm!*

```
public class Came { /* This class is not complete */
   protected int cAmount;
   protected int players;
   protected Deck cards;
   protected Player [] cardPlayer;
   protected Rules rules;

   public void join(Player p}{ /*new player joins the came*/
      cardPlayer[players]=p;
      players++;
   )
   public void round () { /* one deal */
       int pot=0; int i=0;
       cards.suffle();
       for (int i=1;i++;i<players) {
          cardPlayer[i].takeCards(cards.giveCards(cAmount));
          pot += cardPlayer.bet(rules,players);
       }
       Player winner= determineWinner();
       winner.takePot(pot);
    }
    protected Player determineWinner() { /* fids out the winner*/
       int w=0;
       Deck best=cardPlayer[0].getHand();
       for (int i=1;i++;i<players) {
          Deck hand =cardPlayer[i].getHand();
          if (rules.isBetter(best, hand)) {
            w=i; best=hand;
          }
       }
       return cardPlayer[w];
     }
}

public interface Rules {
  public boolean isBetter(Deck bestSoFar; Deck newDeck);
     // returns true if newDeck is better
  public float valueOfDeck(Deck theDeck);
     // computes the ranking
}

public class Player {
   String name;
   Deck hand;
   int money;
   Profile myprofile;
   public Player(String pName, int pMoney, Profile pProfile) {
       name=pName; money=pMoney; hand=null; myProfile=pProfile;
   }
   public int bet(Rules r,int players){
      /* uses rules to evaluate how good the hand is.
         Uses also services of myprofile to decide the bet */
       int toBet=0;
       toBet= myprofile.determineBet(players,money);
       money -=toBet;
       return toBet;
   }
   public void takePot(int pot) {money +=pot};
      /* wins payed */
```

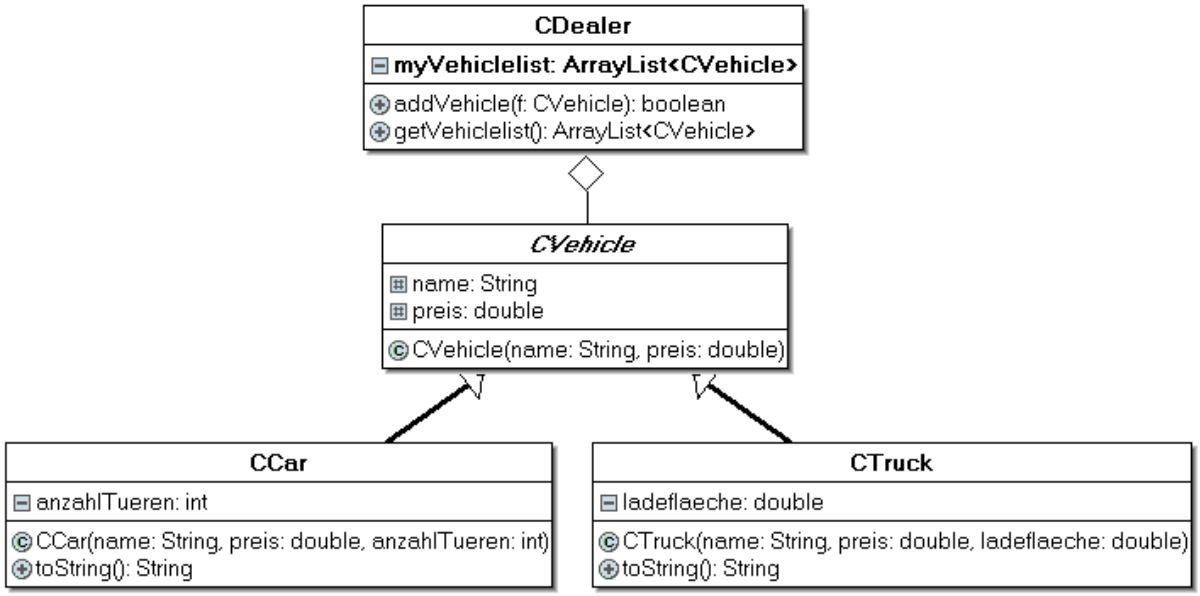
```
    public void takeCards (Deck newcards) {hand=newcards};
        /* new deal  */
    public Deck getHand () {return hand}
        /* show the cards */
}

public class Deck {
    int top=0;
    Card [] cards;
    public void suffle() { /* re-organizes cards */ };
    public Deck giveCards (int number) {
        // creates a new deck with given number of cards taken from
        // the top of deck.  Moves top.
    }
}
interface Profile {
    /* playing strategy */
    int determineBet(int players, int money);
    /* determines the bet */
}
```

*(7.)*

# ***UML - Realisierung von Assoziationen - Lösungen***

*(1.) -(2.): Lösung in Dateiform*

*(3.) (a.)*




*(b.) Lösung in Dateiform*

*(4.), (5.) Lösung in Dateiform*

*(6.)*