JAVA-ÜBUNGSAUFGABEN ASSOZIATIONEN 1

1) Auto - Fahrer (1:n) unidirektional

Implementieren Sie ein Programm mit folgenden Eigenschaften (E):
(Keine Felder, sondern die Klasse ArrayList verwenden).

E1)
Zu einem Auto gehören mehrere Fahrer (1 : * Assoziation)
(d.h. zu einem Auto kann es auch gar keinen Fahrer geben)

E2)
unidirektional (mit Navigierbarkeit in genau eine Richtung): Auto --> Fahrer
wobei Auto das Attribut seineFahrer (Typ: ArrayList) besitzt.

E3)
Ein Auto ist durch sein KFZ-Zeichen (Attribut kfz : String) und ein Fahrer durch seinen Namen (Attribut name : String) charakerisiert.

E4)
Die Fahrerliste eines Autos muss im Konstruktor erzeugt (aber nicht befüllt) werden.

Folgendes muss implementiert werden:
a)
Erstellen Sie die Klassen Auto mit o.g. Eigenschaften und die Klasse Fahrer.
Falls ein Fahrer in der Fahrerliste eines Autos schon vorkommt, darf er nicht nochmals darin aufgenommen werden. Aus der Fahrerliste eines Autos dürfen auch Fahrer entnommen (gelöscht) werden.
Implementieren Sie dazu die entsprechenden Methoden wie z.B. addFahrer(…) und removeFahrer(…). Man könnte sie auch verlinke(…) bzw. entlinke(…) nennen.

b)
In main() müssen die zwei Listen (ArrayList) alleFahrer und alleAutos angelegt und mit folgenden Testdaten befüllt werden:
Die Auto müssen die KFZ-Zeichen a0, a1, a2, a3 und die Fahrer die Namen f0, f1, f2, f3 haben.

c)
Implementieren Sie außerhalb von main() die static-Methode printVerlinkung(…), die zu jedem Auto der Liste alleAutos die zugehörigen Fahrer aus der Liste alleFahrer auf dem Bildschirm ausgibt.

d)
Die folgenden Beziehungen müssen in main() nacheinander implementiert werden:
Dabei muß (der besseren Überprüfung wegen, also zur Kontrolle) jeweils (vor und nach dem Anlegen einer neuen Beziehung) Folgendes beachtet werden:
Die Beziehung, die neu angelegt bzw. hinzugefügt werden soll, muss auf dem Bildschirm angezeigt (ausgegeben) werden. Nach dem Anlegen der Beziehung müssen mit der Methode printVerlinkung(…) alle Verlinkungen auf dem Bildschirm ausgegeben werden.

+ bedeutet hinzufügen und - bedeutet entfernen einer Beziehung.

+ (a1,f1)
+ (a1,f1) // wird nicht zugelassen
+ (a1,f2) // also insgesamt: (1,1); (1,2)
+ (a1,f2) // wird nicht zugelassen
+ (a1,f3) // also insgesamt: (1,1); (1,2); (1,3)
+ (a1,f3) // wird nicht zugelassen
+ (a2,f1) // also insgesamt: (1,1); (1,2); (1,3); (2,1)
+ (a2,f1) // wird nicht zugelassen
+ (a2,f2) // also insgesamt: (1,1); (1,2); (1,3); (2,1); (2,2)
+ (a2,f2) // wird nicht zugelassen
+ (a3,f0) // also insgesamt: (1,1); (1,2); (1,3); (2,1); (2,2); (3,0)
+ (a3,f0) // wird nicht zugelassen
+ (a3,f1) // also insgesamt: (1,1); (1,2); (1,3); (2,1); (2,2); (3,0); (3,1)
+ (a3,f1) // wird nicht zugelassen
- (a1,f1) // also insgesamt: (1,2); (1,3); (2,1); (2,2); (3,0); (3,1)
- (a1,f2) // also insgesamt: (1,3); (2,1); (2,2); (3,0); (3,1)

e)
Im letzten Aufgabenteil kann es zu einem Fahrer mehrere Autos geben.
Das darf aber eigentlich nicht sein, da es zu jedem Fahrer maximal ein Auto geben kann.
Implementieren Sie einen entsprechenden Programmcode in main() bzw. die static-Methode

```
public static void verlinkung(Auto a, Fahrer f,
                              ArrayList<Auto> alleAutos,
                              ArrayList<Fahrer> alleFahrer)
```

Diese Methode darf nur Verlinkungen durchführen, die verhindern, daß es zu einem Fahrer mehrere Autos geben kann.
Implementieren Sie jetzt nochmals den vorigen Aufgabenteil.

f)
Lösen Sie die Aufgabe für eine 1 : 1..* Assoziation.
Das bedeutet, dass zu einem Auto mindestens ein Fahrer existiert.
Dieser muss im Konstruktor als Parameter übergeben werden.

2) Auto - Fahrer (1:n) unidirektional

Implementieren Sie das gleiche Programm wie in der vorigen Aufgabe, nur muss jetzt Folgendes gelten:

a)
Nur alle Autos (nicht die Fahrer) werden in den Feldern alleAutos in main() abgelegt.

b)
Die Fahrer-Objekte werden in der Methode addFahrer(String fname) der Klasse Auto erzeugt.
Statt addFahrer(String fname) könnte man auch den Namen verlinke(String fname) verwenden.

3) Auto - Fahrer (1:n) bidirektional

Implementieren Sie ein Programm mit folgenden Eigenschaften (E):
(Keine Felder, sondern die Klasse ArrayList verwenden).

E1)
Zu einem Auto gehören mehrere Fahrer (1 : * Assoziation) und
zu einem Fahrer gehört maximal ein Auto.

E2)
bidirektional (mit Navigierbarkeit in zwei Richtungen): Auto <--> Fahrer
wobei Auto das Attribut seineFahrer (Typ: ArrayList) besitzt.

E3)
Alle Autos und alle Fahrer werden in den Listen (Klasse ArrayList verwenden) alleAutos und alleFahrer in main() abgelegt.

E4) Die Fahrerliste eines Autos muss im Konstruktor erzeugt (aber nicht befüllt) werden.

E5)
Datenkonsistenz muß garantiert werden, d.h. es müssen folgende 2 Bedingungen erfüllt werden:
==> Es darf kein Fahrer in die Fahrerliste eines Autos aufgenommen werden, der dort schon einmal vorkommt.
==> Außerdem darf kein Fahrer in die Fahrerliste eines Autos aufgenommen werden, so dass dieser Fahrer dann mehrere Autos hat (weil zu einem Fahrer maximal ein Auto gehören darf)

Beispiel:
In main werden in einer Autoliste genau die 3 Autos a1, a2, a3, erzeugt und
in einer Fahrerliste genau die 3 Fahrer f1, f2, f3.
Dann werden folgende Verlinkungen erzeugt:

```
a1 +--------> f1
   |
   |
   +--------> f2
   |
   |
   +--------> f3
```

Die folgenden weiteren Verlinkungen können nicht durchgeführt werden:
(Der obere Graph ist in diesem Sinne schon vollständig!)
+ (a1 , f1) weil (a1,f1) schon existiert
+ (a2 , f3) weil sonst f2 --> {a1, a2} d.h. ein Fahrer --> mehrere Autos
+ (f2 , a1) weil (a1,f1) schon existiert
Es kann also keine einzige weitere Verlinkung mehr erzeugt werden.

Bemerkung:
+ (f2 , a1) bedeutet, dass die Verlinkung vom Fahrer aus erfolgt.

a)
Erstellen Sie die Klassen Auto mit o.g. Eigenschaften und die Klasse Fahrer.
Falls ein Fahrer in der Fahrerliste eines Autos schon vorkommt, darf er nicht nochmals darin aufgenommen werden. Aus der Fahrerliste eines Autos dürfen auch Fahrer entnommen (gelöscht) werden.
Implementieren Sie dazu die entsprechenden Methoden wie z.B. addFahrer(…) und removeFahrer(…). Man könnte sie auch verlinke(…) bzw. entlinke(…) nennen.
Die Methode verlinke(…) und entlinke(…) muss es auch in der Klasse Fahrer geben.
Implementieren Sie außerhalb von main() die static-Methode printVerlinkung(…), die die zu jedem Auto der Liste alleAutos die zugehörigen Fahrer aus der Liste alleFahrer auf dem Bildschirm ausgibt.

b)
In main() müssen zwei Listen (ArrayList) angelegt werden: alleFahrer und alleAutos.
Die darin enthaltenen Autos bzw Fahrer sollen der Einfachheit halber mit a0, a1, a2, a3 und f0, f1, f2, f3 bezeichnet werden.
Die folgenden Beziehungen müssen in main() nacheinander implementiert werden:
Nach jeder eingefügten Beziehung muss (zur Kontrolle) die Methode printVerlinkungAuto(…) aufgerufen werden.
Die Methode printVerlinkungFahrer(…) muss vor dem Programmende einmal aufgerufen werden. Dort müssen dann zu jedem Fahrer die zugehörigen Autos auf dem Bildschirm ausgegeben werden.

```
+ (a1,f1)      // also insgesamt:  (a1,f1)
+ (a1,f1)      // wird nicht zugelassen
+ (f2,a1)      // also insgesamt:  (a1,f1); (a1,f2)
+ (a1,f2)      // wird nicht zugelassen
+ (a1,f3)      // also insgesamt: (a1,f1); (a1,f2); (a1,f3)
+ (a1,f3)      // wird nicht zugelassen
+ (a2,f1)      // wird nicht zugelassen
+ (a2,f2)      // wird nicht zugelassen
+ (a3,f0)      // also insgesamt: (a1,f1); (a1,f2); (a1,f3); (a3,f0);
+ (a3,f0)      // wird nicht zugelassen
+ (a3,f1)      // wird nicht zugelassen
- (a1,f1)      // also insgesamt: (a1,f2); (a1,f3); (a3,f0);
- (f2 a1)      // also insgesamt: (a1,f3); (a3,f0);
```

c)
Lösen Sie die Aufgabe für eine bidirektionale 1 : 1..* Assoziation
Das bedeutet, dass zu einem Auto mindestens ein Fahrer existiert.
Dieser muss im Konstruktor als Parameter übergeben werden.

4) Auto - Fahrer (n:m) bidirektional

Übertragen Sie die vorige Aufgabe auf eine bidirektionale n : m Assoziation.

5) Modellierung einer Bücherei

Eine Bücherei besitzt viele Bücher, deren Bestand sich im Laufe der Zeit verändern kann. Die Bücherei besitzt außerdem einen registrierten Personenkreis (Kunden), der sich natürlich auch verändern kann.
a) In welchem Verhältnis steht die Bücherei zu den Büchern und ausleihenden Personen (Kunden)? Erstellen Sie ein UML mit den Klassen: Bücherei, Kunde, Buch

b) Ein Kunde kann mehrere Bücher ausleihen, aber ein Buch kann nicht gleichzeitig von mehreren Kunden ausgeliehen werden.
b1)
In der Klasse Bücherei müssen folgende Attribute angelegt werden:
- der Name der Bücherei
- die zwei Listen (ArrayList) alleBücher und alleKunden

b2)
Erzeugen Sie in main() ein Objekt (Variable stadtbücherei) der Klasse Bücherei.
Zu Testzwecken soll in diese Bücherei mit folgenden Testdaten befüllt werden:
Die Bücher müssen die Titel b0, b1, b2, b3 und die Kunden die Namen k0, k1, k2, k3 haben.
Zu Testzwecken soll außerdem versucht werden, dass jeder Kunde alle Bücher ausleiht und jedes Buch mehrere Ausleiher haben soll:

---------- Szenario ----------
(k0,b0), (k0,b1), (k0,b2), (k0,b3),
(k1,b0), (k1,b1), (k1,b2), (k1,b3),
....
(k3,b0), (k3,b1), (k3,b2), (k3,b3),
und
(b0,k0), (b0,k1), (b0,k2), (b0,k3),
(b1,k0), (b1,k1), (b1,k2), (b1,k3),
...
(b3,k0), (b3,k1), (b3,k2), (b3,k3),

Dies soll durch die folgenden Methoden der Klasse Bücherei veranlasst werden:

```
public void verlinke(Kunde k, Buch b)
public void verlinke(Buch b, Kunde k)
```

Die Methoden müssen natürlich so intelligent programmiert werden, dass im obigen Szenario keine Inkonsistenzen umgesetzt werden dürfen, wie z.B:
Die folgenden Einträge dürfen gemacht werden
(k0,b0), (k0,b1), (k0,b2), (k0,b3),
aber das Folgende muß verhindert werden:
(k1,b0), (k1,b1), (k1,b2), (k1,b3),
denn (k1,b0) bedeutet, daß Buch b0 zweimal gleichzeitig an die 2 Kunden k0 und k1 ausgeliehen wird.
Implementieren Sie diesen Fall.

b3)
Mit Hilfe der folgenden Methoden der Klasse Bücherei sollen alle Verlinkungen entfernt werden:

```
public void entlinke(Kunde k, Buch b)
public void entlinke(Buch b)
```

c) Ein Kunde kann mehrere Bücher ausleihen und im Laufe der Zeit kann ein Buch von mehreren Kunden ausgeliehen worden sein. Aus statistischen Gründen soll gespeichert werden, ob ein Buch zu einem Kunden gehört(e).
Es soll also feststellbar werden, ob ein Buch zu einem Kunden gehört (d.h. ob es irgendwann einmal von ihm ausgeliehen wurde). Aus datenschutzrechtlichen Gründen soll dieses Datum nicht gespeichert bzw. festgehalten werden.
Modellieren und implementieren Sie diesen Fall.

d) Im Gegensatz zur letzten Teilaufgabe soll feststellbar sein, wann ein Buch von einem Kunden ausgeliehen wurde.
Modellieren und implementieren Sie diesen Fall.

6) verschiedene Beziehungen modellieren

Modellieren Sie die folgenden Beziehungen:
Mensch - Kopf

Gebäude - Zimmer , Treppen

Flugzeug - Piloten

Karosserie - Fahrwerk, Flügel

Flügel - Triebwerk

7) Modellierung eines Fuhrparks

Die modellierte Autovermietung bietet verschiedene Fahrzeugtypen als Mietfahrzeuge an. die wichtigsten Daten, die für alle Fahrzeuge erfasst werden sind Hersteller, Kilometerstand und Baujahr.
Die Personenkraftwagen und Wohnmobile verfügen über eine festgelegte Personenzahl, die Lastwagen über ein max. zulässiges Gesamtgewicht.
Die Gesamtkosten des Fuhrparks setzen sich zusammen aus laufenden Kosten und Fixkosten, wie z.B. Kosten für Kfz-Versicherungen und Reparaturen.
Für jedes Element (Fahrzeug) des Fuhrparks müssen ein Belegplan und die Gesamtkosten erfasst werden, ebenso die Fahrzeugdaten.
Der Fuhrpark verfügt über eine sich permanent ändernde Anzahl von Fahrzeugen.
Zu allen einzugebenden/auszulesenden Daten müssen zusätzlich noch Dialoge erstellt werden (Dialogklassen), mittels der die Daten der einzelnen Klassen (über Tastatur verändert werden können).