JAVA-ÜBUNGSAUFGABEN DATENBANK 1

1)

Eine Kuh soll durch ihren Namen und das Volumen der von ihr täglich produzierten Milchmenge modelliert werden.
Es müssen verschiedene Kühe erstellt und in der Datenbank (DB) dbkuehe abgespeichert werden.

a)

Erstellen Sie in Java die Klasse Kuh (Attribut name: String, Attribut volumen : double)
Erzeugen Sie mit Hilfe einer Schleife die 5 Kühe (name, anzahl) :
(k0, 100), (k1, 101), (k2, 102), (k3, 103), (k4, 104)
Diese sollen in der ArrayList kuhListe mit Datentyp Kuh abgespeichert werden.
Schreiben Sie die Methode

```
public static void printKuhArrayList(ArrayList<Kuh> pKuhListe)
```

die die Kühe der ArrayList mit den Werten ihrer Attribute auf dem Bildschirm ausgibt.

b)

Erstellen Sie die Datenbank dbkuehe mit der Tabelle tabkuh (name, volumen), wobei name ein Primärschlüssel sein muss und volumen das Volumen (Datentyp double) der täglich erzeugten Milchmenge einer Kuh.
Speichern Sie darin die oben erstellten Kühe ab.
Schreiben Sie die Methode printDB(…), die die Kühe der DB mit den Werten ihrer Attribute auf dem Bildschirm ausgibt.
Um die Zugriffe auf die DB überichtlicher zu gestalten, wird empfohlen, die Klasse DBZugriff mit den folgenden Methoden zu implementieren:

```
public void openDB() throws Throwable
public void closeDB() throws SQLException
```

In den folgenden Methoden muß pSQL die SQL-Zeichenfolge sein, mit der auf die DB zugegriffen wird.

```
public ResultSet select(String pSQL) throws SQLException
public void insert(String pSQL) throws SQLException
public void update(String pSQL) throws SQLException
public void delete(String pSQL) throws SQLException
```

Bemerkung:
Falls man in der Klasse DBZugriff eine (selbstsprechende) Variable statementSQL deklarieren will, gibt es 2 Möglichkeiten dies zu tun: lokal in den Methoden oder als Attribut (global).
Vorschlag: statementSQL nur global für openDB(), closeDB() bneutzen.
In den restlichen Methoden diese Variable lokal deklarieren.
Sonst gibt es Probeme mit der Methode next(): diese wird sonst nämlich geschlossen
Siehe dazu die Doku zu ResultSet:
Note: A ResultSet object is automatically closed by the Statement object
that generated it when that Statement object is closed, re-executed,
or is used to retrieve the next result from a sequence of multiple results.

c)
Um die Lösung noch professioneller zu gestalten soll ab hier Folgendes gemacht werden:
Erstellen Sie die zusätzlich noch die Klasse DBManager mit den Methoden:

- c1)
Schreibt ein Tier in die DB. Existiert das Tier schon (Primärschlüssel name), wird es upgedatet, ansonsten neu eingefügt. Zum Einfügen und Updaten werden die unten aufgeführten private-Methoden verwendet.
```
public void writeRecord(String tabelle, Kuh pKuh) throws
Throwable
```

- c2)
Fügt ein Tier in die DB ein.
Falls es schon existiert muß ein Fehlercode zurückgeliefert werden.
```
private int insertRecord(String tabelle, Kuh pKuh) throws
Throwable
```

- c3)
Updatet ein schon vorhandenes Tier der DB.
Falls es noch nicht existiert, muß ein Fehlercode zurückgeliefert werden.
```
private void updateRecord(String tabelle, Kuh pKuh) throws
Throwable
```

- c4)
Liest eine Kuh (deren Namen in pKuh steht) aus der Tabelle tabkuh
```
public void readKuh(Tier pKuh) throws Throwable
```

- c5)
Löscht alle Einträge der DB
```
public void deleteDB() throws Throwable
```

Gibt alle Einträge der DB auf dem Bildschirm aus.
```
public void printDB() throws Throwable
```

d)
Lesen Sie alle Daten aus der DB aus, zählen jeweils den Wert des Volumens um 1 hoch und schreiben diese geänderte Daten mit der Methode writeRecord wieder in die DB zurück.

e)
Schreiben Sie zusätzlich noch mit Hilfe einer Schleife die folgenden 5 Kühe mit der Methode writeRecord(..) in die DB.
(k5, 500), (k6, 600), (k7, 700), (k8, 800), (k9, 900)

2)
Die Klassen Kuh , Henne erben von der Klasse Tier.
Von einer Henne interessiert die Anzahl der Eier pro Tag. Von einer Kuh dagegen das Volumen der Milch, die jeden Tag von ihr gemolken wird.
Es müssen verschiedene Kühe und Hennen erstellt und in der Datenbank (DB) tiereDB abgespeichert werden.

a)
Erstellen Sie in Java die Klassen Tier (Attribut name: String), Henne(Attribut anzahl : int), Kuh (Attribut volumen : double)
Erzeugen Sie 5 Kühe: (name, volumen):
(k0, 100), (k1, 101), (k2, 102), (k3, 103), (k4, 104)
und 5 Hennen: (name, anzahl):
(h0, 0), (h1, 1), (h2, 2), (h3, 3), (h4, 4)
Diese sollen in genau einer gemeinsamen ArrayList mit Datentyp Tier abgespeichert werden.
Schreiben Sie die Methode printTierArrayList(…), die die Tiere der ArrayList mit den Werten ihrer Attribute auf dem Bildschirm ausgibt.

b)
Erstellen Sie die DB dbtiere mit den 2 Tabellen tabkuh (name, volumen) und tabhenne(name, anzahl), wobei jeweils name ein Primärschlüssel sein muss.
Speichern Sie darin die oben erstellten Kühe und Hennen ab.
Implementieren Sie die Methode printDBTiere (…), die die Tiere der DB mit den Werten ihrer Attribute auf dem Bildschirm ausgibt.

c)
Um die Lösung noch professioneller zu gestalten soll ab hier Folgendes gemacht werden:
Erstellen Sie die zusätzlich noch die Klasse DBManager mit den Methoden:

- c1)
Schreibt ein Tier in die DB. Existiert das Tier schon (Primärschlüssel name), wird es upgedatet, ansonsten neu eingefügt. Zum einfügen und updaten werden die unten aufgeführten private-Methoden verwendet.
```
public void writeRecord(String tabelle, Tier pTier) throws
Throwable
```

- c2)
Fügt ein Tier in die DB ein.
Falls es schon existiert muß ein Fehlercode zurückgeliefert werden.
```
private int insertRecord(String tabelle, Tier pTier) throws
Throwable
```

- c3)
Updatet ein schon vorhandenes Tier der DB.
Falls es noch nicht existiert, muß ein Fehlercode zurückgeliefert werden.
private void updateRecord(String tabelle, Tier pTier) throws Throwable

- c4)
Liest eine Henne (deren Namen in pTier steht) aus der Tabelle tabkuh
public void readHenne(Tier pTier) throws Throwable

- c5)
Liest eine Kuh (deren Namen in pTier steht) aus der Tabelle tabkuh
public void readKuh(Tier pTier) throws Throwable

- c6)
Löscht alle Einträge der DB
public void deleteDB() throws Throwable

- c7)
Gibt alle Einträge der DB auf dem Bildschirm aus.
public void printDB() throws Throwable

d)
Lesen Sie alle alle Daten aus der DB aus, zählen jeweils den Wert der Anzahl bzw. des Volumens um 1 hoch und schreiben diese geänderte Daten mit der Methode writeRecord(...) wieder in die DB zurück.

e)
Schreiben Sie zusätzlich noch mit Hilfe einer Schleife die folgenden 5 Kühe und Hennen mit der Methode writeRecord(..) in die DB.
(k5, 500), (k6, 600), (k7, 700), (k8, 800), (k9, 900)
(h5, 5), (h6, 6), (h7, 7), (h8, 8), (h9, 9)

3)
Die Beziehung zwischen einem Auto und einem Fahrer wurde in einer früheren Übung durch eine Assoziation (dargestellt durch UML) realisiert (z.B. 1:n bzw. n:m)
Realsieren Sie dies jetzt durch eine Datenbank.

4)
Erzeugen Sie mit Hilfe von MySQL die Datenbank "dBZufall" und die 2 einspaltigen Integer-Tabellen "Muenze" und "Wuerfel".
a)
Es sollen mit Hilfe eines echten Würfels (oder mit selbst ausgedachten Zahlen) z.B. das Ergebnis dreier Würfe in der MySQL-Datenbank "dBZufall" in der einspaltigen Tabelle "Wuerfel" gespeichert werden. Konkret: die Zahlen 3, 1, 6.

b) Machen Sie die Probe:
Lesen Sie diese Zahlen aus der Tabelle "Wuerfel" aus und geben Sie diese auf dem Bildschirm aus.

c)
Löschen Sie die Inhalte (also alle Zahlen) der Tabellen "Muenze" und "Wuerfel".

d)
Es sollen mit Hilfe eines simulierten Würfels eine bestimmte Anzahl Zahlen (z.B. 50) erwürfelt werden und (mit Hilfe einer Schleife) in der einspaltigen Tabelle "Wuerfel" der MySQL-Datenbank "dbZufall" gespeichert werden.

e) Machen Sie die Probe:
Lesen Sie diese Zahlen aus der Tabelle "Wuerfel" aus und geben Sie diese auf dem Bildschirm aus.

f)
Dann sollen aus der Tabelle "Wuerfel" alle Münzzahlen (das sind die Zahlen 1 und 2) ausgelesen werden und in der Tabelle "Muenze" abgespeichert werden.
Lesen Sie dazu mit einem geeigneten SQL-Befehl alle Münzzahlen aus der Tabelle "Wuerfel" aus und schreiben diese dann mit Hilfe einer Schleife in die Tabelle "Muenze".

g) Machen Sie die Probe:
Lesen Sie diese Zahlen aus der Tabelle "Muenze" aus und geben Sie diese auf dem Bildschirm aus.

h) schwierig
machen Sie die letzte Teilaufgabe mit genau einem SQL Befehl.
Recherchieren Sie dazu die 2 möglichen Anwendungen des SQL-Befehls INSERT INTO

i)
Machen Sie die ganze Aufgabe mit den OOP - Richtlinien, also mit Klassen und Methoden.

5)
a)
Es sollen die "wunderbaren Zahlen" durch Expansion entwickelt werden:
(analog dem Universum, das sich aus dem Urknall immer weiter expandiert hat)
Aus den bestehenden Zahlen werden immer wieder nach einer bestimmten Vorschrift neue gebildet und in eine MySQL-Datenbank (DB) eingefügt (aber nur wenn sie nicht darin schon enthalten sind).
Vorschrift:
Nimm jeweils 2 verschiedene Zahlen, bilde deren Summe und füge sie in die DB ein, aber nur, wenn diese nicht schon in der DB enthalten sind.
Man beginnt z.B. nach dem Urknall mit den Atomen 3 und 5:
{3 ; 5}
1. Expansion: {3 ; 5 ; 8}
2. Expansion: {3 ; 5 ; 8 ; 11 ; 13}
3. Expansion: {3 ; 5 ; 8 ; 11 ; 13 ; 14; 16 ; 18 ; 19 ; 21 ; 24}
...
Wenn man "unendlich" oft expandiert, bekommt man die Menge der wunderbaren Zahlen.

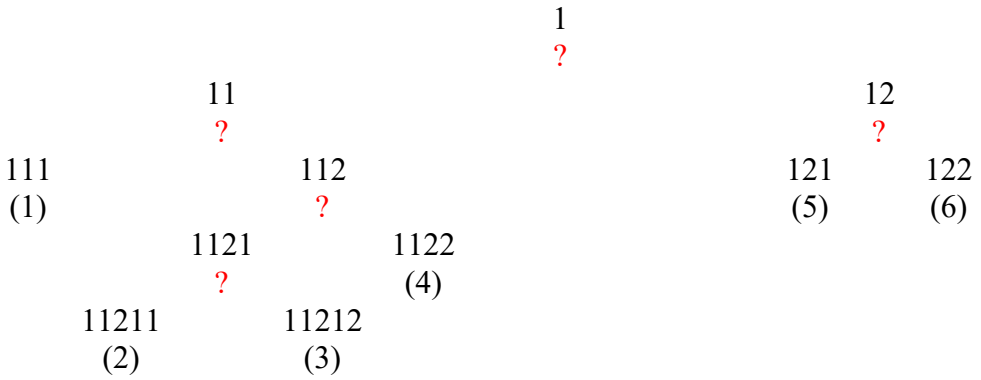
Bemerkung:
Man könnte diese "wunderbaren Zahlen" auch in eine Datenbank, in ein (genügend großes) Feld oder in eine verkettete Liste schreiben.

b)
Wenn genügend oft expandiert wird, ist es möglich, daß ab einer bestimmten Zahl die Zahlen keine Lücken mehr haben. Es könnte z.B. sein, daß ab der Zahl 53 die neu hinzukommenden Zahlen 54, 55, 56, 57, 58, 59, ... sind.
Dann könnte man diese Zahl 54 die Barkon-Konstante nennen, also wäre dann:
Barkon(3,5) = 54
Bestimmen Sie die Barkon-Kostante bei einem Universum, das mit einem anderen Urknall beginnt z.B. mit 13 und 17, also Barkon(13,17).

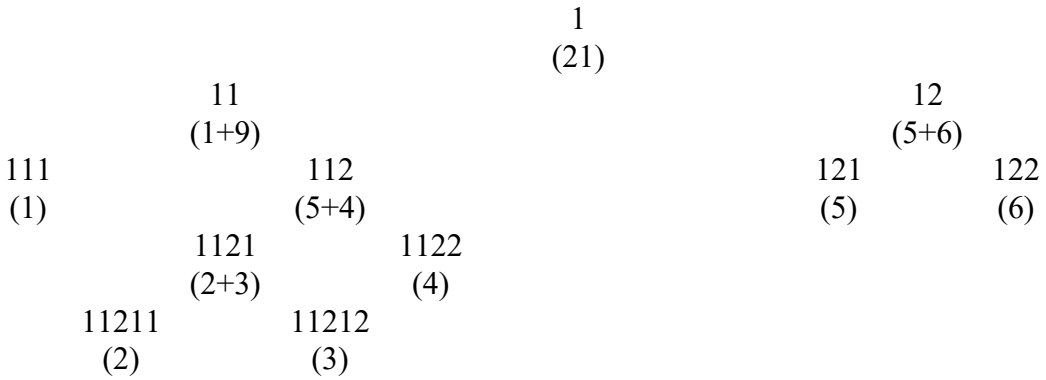
c)
Man könnte in der Tabelle nicht nur die Zahl speichern, sondern zusätzlich noch 2 Zahlen, aus denen die Zahl gebastelt wurde.
also z.B.
3, -1, -1 (weil 3 ein Atom ist)
5, -1, -1 (weil 5 ein Atom ist)
3, 5, 8 (weil 8 aus 3 und 5 gebastelt wurde)
....

6)
Eine Elektrofirma verkauft verschiedene Geräte, die jeweils zu einem bestimmten Preis verkauft werden.
Für jedes Gerät gibt es genau die 2 folgenden verschiedenen Möglichkeiten:
M1) Es ist ein elementares Gerät (Atom), das aus keinem weiteren Gerät dieser Firm besteht.
M2) Es ist ein verschachteltes Gerät, das aus Gründen der Einfachheit aus genau 2 weitern Geräten dieser Firm besteht.
Die Preise (integer) der elementaren Geräte sind fest vorgegeben und stehen in dem Beispiel unter den Gerätenummern (integer).
Die Preise der verschachtelten Geräte werden aus den einfachen Geräten berechnet.
Man kann diese Sitaution auch durch einen Baum beschreiben:

Beispiel:



Berechnung der Preise der Geräte (Knoten des Baums)



Die Geräte werden in einer Datenbank gespeichert, konkret in der Tabelle Geraet1. Die Tabelle Geraet1 besteht aus folgenden Spalten:
G G1 G2 Preis
wobei G die (positiven) Gerätenummer und G1 und G2 die Gerätenummern der sich darin befindlichen 2 weiteren Geräte bedeutet. Mit P wir der Preis des Gerätes bezeichnet.
Ist ein Gerät ein elementares Gerät, müsen G1 und G2 den Wert -1 haben.
Der Preis ein verschachtelten Geräts muß standardmäßig am Anfang -1 sein.
Er wird später durch ein Java-Programm (mit SQL-Befehlen) berechnet und in die Tabelle eingetragen.

Beispiel

G (int)	G1 (int)	G2 (int)	P (int)
1	11	12	-1
11	111	112	-1
12	121	123	-1
111	-1	-1	1
112	1121	1122	-1
121	-1	-1	5
122	-1	-1	6
1121	11211	11212	-1
1122	-1	-1	4
11211	-1	-1	2
11212	-1	-1	3

Die elementaren Geräte haben also die Nummern
111, 121, 122, 1122, 11211, 11212
Durch Summenbildung werden die Preise der darüberliegenden Knoten
berechnet.
Durch Berechnung ergibt sich für das Gerät mit der Gerätenummer 1 einen Verkaufspreis von 21 Euro.

a)
Erstellen Sie die MySQL-Datenbank dBGeraet1.
Implementieren Sie die 4-parametrige Methode insertGerät(...), die einen Datenmsatz in die Tabelle Geraet1 einfügt.
Fügen Sie mit Hilfe dieser Methode die obigen Datensätze des Beispiels (mit noch unberechneten Endpreisen der verschachtelten Geräte) in die Tabelle ein.

b)
Implementieren Sie die Methode löscheBaumDB (...), die alle Datenmsätze der Datenbank löscht.

c)
Schreiben Sie eine Methode, die die noch unberechneten Endpreisen der verschachtelten Geräte berechnet und die Tabelle aktualisert (mit UPDATE).
Tipp:
Machen Sie zuerst eine entsprechenden SELECT -Befehl und benutzen Sie dann einen UPDATE-Befehl.

7)
Siehe auch: https://www.mpg.de/425007/pdf.pdf

Es soll eine Telefonkette erstellt werden. In dieser sind die Telefonnummern der nächsten anzurufenden Personen gespeichert.
Dies muß mit Hilfe einer MySQL-Datenbank realisiert werden.

Beispiel: Tabelle "verketteteListe"

Tnr (int)	TnrNext (int)	Name (String)	Delay (double)
1	2	a1	10
2	3	a2	20
3	4	a3	30
4	5	a4	40
5	6	a5	50
6	7	a6	60
7	8	a7	70
8	9	a8	80
9	10	a9	90
10	11	a10	100
11	-1	a11	110

Tnr: Telefonnummer
TnrNext: Telefonnummer der als nächstes anzurufenden Person. Das Ende der verketteten Liste wird durch -1 gekennzeichnet.
Name: Name der anrufenden Person.
Delay: Zeit die Person vertreichen läßt, bis sie die nächste Person anruft (Verzögerung).
Wenn man alle "Delays" der Telefonkette aufsummiert, bekommt man die Totzeit der Telefonkette mit, d.h. die Zeit, die vom 1. Anruf bis zum letzten Anruf verstreicht.

Bemerkung:
Man könnte noch die Anzahl speichern, wie oft die Telefonkette insgesamt genutzt wurde.

Um nicht jedesmal den Anfang und das Ende der verketteten Liste berechnen zu müssen, werden diese Werte in der Tabelle "tabellenInfos" gespeichert (die nur aus einem Datensatz besteht) und das Listenende bei Erweiterung der verketteten Liste aktualisiert.

Beispiel: Tabelle "tabellenInfos"

Anfang (int)	Ende (int)
1	11

Dieses Beispiel bezieht sich auf obige Tabelle "verketteteListe".

a)
Erstellen Sie die MySQL-Datenbank "dBTelefonkette".
Erstellen Sie die Tabelle "verketteteListe" mit den folgenden Einträgen:
Implementieren Sie die 4-parametrige Methode insertDatensatz (...), die einen Datensatz in die Tabelle "verketteteListe" einfügt.
Fügen Sie mit Hilfe dieser Methode die obigen Datensätze des Beispiels in die Tabelle ein.

b)
Implementieren Sie die Methode printDB(...), die alle Datensätze der verketteten Liste (und die Telefonnumer des Listenanfangs und Listenendes) auf dem Bildschirm ausgibt.

c)
Implementieren Sie die Methode löscheTelefonketteDB(), die alle Datensätze der Datenbank löscht.

d)
Implementieren Sie die Methode berechneTotzeit(...), die die Totzeit der Telefonliste berechnet.

e)
Wenn man mit Hilfe der Methode insertRecord(...) die Datensätze des obigen Beispiels in die Tabelle "verketteteListe" einfügt, muß der Programmierer aufpassen, daß die Verlinkung stimmt und daß das Listenende immer aktaulisert wird (in der Tabelle "listeninfos").
Implementieren Sie deshalb die Methode appendRecord(int tnr, int name, double delay), die einen Datensatz an das Ende der verketteten Liste anfügt (und die Verlinkung bzw. das Listenende regelt).
Fügen Sie mit Hilfe dieser Methode die obigen Datensätze des Beispiels in die Tabelle ein.

f)
Implementieren Sie die Methode insertLeft (...), die einen Datensatz links einer bestimmten Stelle in der verketten Liste einfügt.
Wenn dies links der Listenanfangs geschieht, ändert sich der Listenanfang.
Implementieren Sie die Methode insertRight (...), die einen Datensatz links einer bestimmten Stelle in der verketten Liste einfügt.
Wenn dies rechts des Listenends geschieht, ändert sich das Listenende.

g)
Erstellen Sie eine doppelt verkettete Liste. so daß die Liste auch von hinten nach vorne durchlaufen werden kann.
Implementieren Sie dazu die entsprechenden Methoden analog zu den vorigen Teilaufgaben.