Programmieraufgabe "Fahrzeugverwaltung"

Programm zur Verwaltung von Fahrzeugen eines Online-Händlers.

1. Abstrakte Klasse *Fahrzeug*

Die abstrakte Klasse *Fahrzeug* dient zum Speichern von Informationen über Fahrzeuge eines Online-Händlers. Dabei sollen folgende Daten als private Instanzvariablen erfasst und notwendige öffentliche Zugriffsmethoden (*set…* und *get…*) erstellt werden:

- *Marke*
- *Modell*
- *Baujahr*
- *Grundpreis*
- *Id* (Fahrzeugnummer - Zahl, eindeutig, aber nicht notwendigerweise fortlaufend)

Wählen Sie passende Datentypen.

Es soll ein Konstruktor implementiert werden, der es ermöglicht, die entsprechenden Instanzvariablen direkt zu setzen. Prüfen Sie, ob es sich um plausible Werte handelt. (z.B., das *Baujahr* darf nicht in der Zukunft liegen, etc.)

Sollten Bedingungen nicht erfüllt sein, werfen Sie eine *IllegalArgumentException* mit einer vorgegebenen Fehlermeldung (siehe Punkt 7).

Die Methode *getAlter()* soll das Alter des Fahrzeuges berechnen.

Zusätzlich soll es eine Methode *getPreis()* geben, die den Preis eines Fahrzeuges auf Basis des Grundpreises minus Rabatt berechnet. Zu diesem Zweck soll eine abstrakte Methode *getRabatt()* in der Klasse Fahrzeug definiert und innerhalb von *getPreis()* verwendet werden.

2. Klassen Pkw und Lkw

Es sollen zwei konkrete (nicht-abstrakte) Unterklassen *Pkw* und *Lkw* von der abstrakten Klasse *Fahrzeug* abgeleitet werden. Die Klasse *Pkw* hat eine zusätzliche private Instanzvariable für das Jahr in dem das letzte Service durchgeführt wurde.

Beide Klassen müssen die Methode *getRabatt()* implementieren: für einen LKW erhält man einen Rabatt, der vom Fahrzeugalter abhängt (pro Jahr einen Rabatt von 5%), während man für einen PKW einen Rabatt abhängig vom Fahrzeugalter und Servicejahr bekommt (für das Fahrzeugalter pro Jahr einen Grundrabatt von 5% plus 2% pro Jahr seit dem letzten Service. Der maximal mögliche Rabatt beträgt für PKW 15% und für LKW 20% des Grundpreises; das Minimum 0%.

Die Methode *toString()* (geerbt von *Object*) soll überschrieben werden, sodass gemäß dem vorgegebenen Format (siehe Punkt 8) alle Fahrzeugdaten als String zurückgegeben werden.

3. Interface *FahrzeugDAO*

Dieses Interface definiert Methoden zum Zugriff auf die Fahrzeugdaten unabhängig von der konkreten Implementierung der persistenten Datenspeicherung. (vgl. *Data Access Object*).

Das Interface *FahrzeugDAO* enthält abstrakte Methoden zum Einlesen und Speichern von Fahrzeugobjekten.

- Die Methode *getFahrzeugList()* gibt alle persistent gespeicherten Fahrzeugobjekte als *java.util.List* zurück.

- Die Methode *getFahrzeugbyId(int ...)* gibt anhand der Fahrzeugnummer ein *Fahrzeug*-Objekt zurück. Falls das Fahrzeug nicht gefunden wird, soll *null* zurückgeben werden.

- Die Methode *speichereFahrzeug(Fahrzeug …)* soll ein Fahrzeugobjekt persistent (in einer Datei) speichern. Stellen Sie sicher, dass beim Speichern eines neuen Fahrzeuges nicht die *Id* eines bereits gespeicherten Fahrzeugs verwendet wird. Werfen Sie in diesem Fall eine *IllegalArgumentException* mit einer entsprechenden Fehlermeldung (siehe Punkt 7).

- Die Methode *loescheFahrzeug(Fahrzeug …)* soll ein bestehendes Fahrzeug von der Datei löschen. Falls es das Fahrzeug nicht gibt, soll eine *IllegalArgumentException* mit entsprechender Fehlermeldung (siehe Punkt 7) geworfen werden.

4. Klasse *SerializedFahrzeugDAO*

Die Klasse *SerializedFahrzeugDAO* implementiert das Interface *FahrzeugDAO*.

Realisieren Sie die persistente Speicherung der Fahrzeugdaten in einer Datei mittels Java *Object Serialization.*

Der Klasse soll im Konstruktor ein String mit dem Namen der Datei übergeben werden.

Bei Fehlern aus Dateioperationen soll eine einzeilige Fehlermeldung, die mit dem String

- "Fehler bei Serialisierung:" oder
- "Fehler bei Deserialisierung:"

beginnt, ausgegeben werden und das Programm mit *System.exit(1)* beendet werden.

5. Klasse *FahrzeugManagement*

Die Klasse *FahrzeugManagement* soll die Business-Logik implementieren. Die Klasse soll eine private Instanzvariable *fahrzeugDAO* (vom Typ *FahrzeugDAO)* besitzen, um auf die Fahrzeugdaten zugreifen zu können.

Implementieren Sie Methoden, die folgende Funktionalität realisieren.

- Alle Daten aller Fahrzeuge bereitstellen
- Alle Daten eines Fahrzeugs bereitstellen
- Neues Fahrzeug hinzufügen
- Bestehendes Fahrzeug löschen
- Gesamtzahl aller Fahrzeuge ermitteln
- Gesamtzahl aller PKWs ermitteln
- Gesamtzahl aller LKWs ermitteln
- Durchschnittspreis aller Fahrzeuge ermitteln
- Id(s) des(r) ältesten Fahrzeugs(e) ermitteln

Anmerkung: Unterscheidung des Fahrzeugtyps mittels *instanceof*-Operator möglich

6. Frontend

Schreiben Sie ein Java-Programm *FahrzeugClient* das unter Verwendung der Klasse *FahrzeugManagement* die nachfolgend beschriebene Kommandozeilenschnittstelle bereitstellt. Achten Sie darauf, dass die Programmausgaben den unten angeführten Beispielen **exakt** entsprechen!

Das Programm soll Aufrufe in folgendem Format unterstützen:

```
java FahrzeugClient <Datei> <Parameter>
```

<Datei>: Name der Datei für die Serialisierung. Falls die Datei nicht existiert soll sie erstellt werden.

<Parameter>: show, add, del, count, meanprice, oldest. Pro Aufruf kann jeweils nur einer dieser Parameter angegeben werden.

- Parameter 'show'
  - Alle Daten aller Fahrzeuge ausgeben

Beispielaufruf:

```
java FahrzeugClient <Datei> show
```

Ausgabe:

```
Typ:          PKW
Id:           5
Marke:        Tesla
Modell:       Model S
Baujahr:      2016
Grundpreis:   65000.00
Servicejahr:  2016
Preis:        60450.00

Typ:          PKW
Id:           2
Marke:        Opel
Modell:       Manta
Baujahr:      1972
Grundpreis:   21000.00
Servicejahr:  2013
Preis:        17850.00

Typ:          LKW
Id:           3
Marke:        MAN
Modell:       TGX 6X2
Baujahr:      2014
Grundpreis:   56763.00
Preis:        48248.55

Typ:          LKW
Id:           22
Marke:        Steyr
Modell:       990
Baujahr:      1972
Grundpreis:   6900.00
Preis:        5520.00
```

- Parameter 'show <id>'
  - Alle Daten eines Fahrzeuges ausgeben

Beispielaufruf:
```
java FahrzeugClient <Datei> show 2
```
Ausgabe:
```
Typ:          PKW
Id:           2
Marke:        Opel
Modell:       Manta
Baujahr:      1972
Grundpreis:   21000.00
Servicejahr: 2013
Preis:        17850.00
```

- Parameter 'add lkw <id> <marke> <modell> <baujahr> <grundpreis>'
  - LKW persistent hinzufügen

Beispielaufruf:
```
java FahrzeugClient <Datei> add lkw 3 MAN "TGX 6X2" 2014 56763
```

- Parameter 'add pkw <id> <marke> <modell> <baujahr> <grundpreis> <ueberpruefungsdatum>'
  - PKW persistent hinzufügen

Beispielaufruf:
```
java FahrzeugClient <Datei> add pkw 5 Tesla "Model S" 2016
    65000 2016
```

- Parameter 'del <id>'
  - Fahrzeug löschen

Beispielaufruf:
```
java FahrzeugClient <Datei> del 3
```

- Parameter 'count'
  - Gesamtzahl der erfassten Fahrzeuge berechnen

Beispielaufruf:
```
java FahrzeugClient <Datei> count
```
Ausgabe:
```
4
```

- Parameter 'count <type>'
  - Gesamtzahl der LKWs berechnen

Beispielaufruf:
```
java FahrzeugClient <Datei> count lkw
```
Ausgabe:
```
2
```

- Parameter 'count <type>'
  - Gesamtzahl der PKWs berechnen

Beispielaufruf:
```
java FahrzeugClient <Datei> count pkw
```
Ausgabe:
```
2
```

- Parameter 'meanprice'
  - Durchschnittspreis aller Fahrzeuge berechnen

Beispielaufruf:
```
java FahrzeugClient <Datei> meanprice
```
Ausgabe:
```
33017.14
```

- Parameter 'oldest'
  - Ältest(e) Fahrzeug(e) suchen

Beispielaufruf:
```
java FahrzeugClient <Datei> oldest
```
Ausgabe:
```
Id: 2
Id: 22
```

7. Fehlermeldungen

Alle auf Grund ungültiger oder fehlerhaften Eingaben geworfenen *Exceptions* müssen abgefangen und das Programm mit einer Fehlermeldungen beendet werden. Es darf dabei immer **nur eine der folgenden Fehlermeldungen** ausgegeben werden:

- "Error: Parameter ungueltig."
- "Error: Baujahr ungueltig."
- "Error: Grundpreis ungueltig."
- "Error: Servicejahr ungueltig."
- "Error: Fahrzeug bereits vorhanden. (id=<id>)"
- "Error: Fahrzeug nicht vorhanden. (id=<id>)"

Beispielaufruf:
```
java FahrzeugClient <Datei> add pkw 7 VW Golf 2005 10000
```
Ausgabe:
```
Error: Parameter ungueltig.
```

Beispielaufruf:
```
java FahrzeugClient <Datei> add pkw x VW Golf 2005 10000 2015
```
Ausgabe:
```
Error: Parameter ungueltig.
```

Beispielaufruf:
```
java FahrzeugClient <Datei> add pkw 7 VW Golf 2020 25000 2017
```
Ausgabe:
```
Error: Baujahr ungueltig.
```

Beispielaufruf:
```
java FahrzeugClient <Datei> del 4711
```
Ausgabe:
```
Error: Fahrzeug nicht vorhanden. (id=4711)
```

8. Achten Sie auch auf folgende Punkte:

- Eingaben
  Falls Parameter Leerzeichen enthalten können Sie " " verwenden (siehe Beispielaufruf 'add').
- Ausgaben
  Gleitkommazahlen immer mit '.' (Punkt) als Dezimaltrenner und genau 2 Stellen hinter dem Komma ausgeben. z.B.: 12.35
  Benutzen Sie dafür z.B. die Methode `Fahrzeug.getDecimalFormat()`

Beispiel:

```
Double preis = 12.345;
DecimalFormat df = Fahrzeug.getDecimalFormat();
System.out.println("Preis:       " + df.format(preis));
```

Ausgabe:

```
Preis:       12.35
```

9. Abgabemodalitäten

Abgabetermin: **Mittwoch, 18.11.2020 12:00** auf der Online-Abgabeplattform

Verwenden Sie als Basis zur Entwicklung Ihres Programms die im Archiv ***Aufgabe1PLC20WS.zip*** enthaltenen Java Klassen/Interfaces. Ändern Sie keinesfalls die Klassennamen oder das Interface *FahrzeugDAO* und belassen Sie alle Files im Default-Package.

Für eine positive Abgabe ist das lauffähige (Java 1.8) Programm mit allen hier genannten Anforderungen **rechtzeitig vor dem Abgabetermin** auf der Online-Plattform abzugeben. Weitere Informationen dazu in den VU-Einheiten/Tutorien und auf Moodle.