Assignment "Vehicle Management"

Implement an application for managing vehicles of a vehicle dealer.

1. Abstract Class `Vehicle`

The abstract class **`Vehicle`** is used to store information about vehicles. The following information should be captured using private instance variables:

- brand
- model
- year built
- base price
- id (unique vehicle number)

A constructor for directly setting these variables must be implemented and plausibility checks performed, e.g., the year built must not be in the future.

In case a plausibility check fails, an **`IllegalArgumentException`** with a predefined error message must be thrown (see Section 7).

The instance method **`getAge()`** for computing the current age must be provided.

The instance method **`getPrice()`** must be provided for computing the price of a vehicle based on the base price minus the discount. For this purpose, an abstract method **`getDiscount()`** must be defined and used within **`getPrice()`**.

2. Classes `Car` and `Truck`

`Car` and **`Truck`** are concrete classes derived from **`Vehicle`**. Class **`Car`** has a private instance variable to store the year of the last inspection.

Both classes implement **`getDiscount()`**. For a car the discount from the base price is computed based on its age and the time passed since its last inspection: for each year of age 5% and 2% for each year after the last inspection. For a truck the discount from the base price is computed based on its age, for each year 5% discount. The maximum possible discount is limited to 15% for cars and 20% for trucks.

3. Interface `VehicleDAO`

This interface specifies abstract methods for storing, retrieving, and deleting vehicles independently of how persistent storage is implemented. (cf. *Data Access Object*).

The interface **`VehicleDAO`** provides the following abstract methods:

- **`getVehicleList()`** returns all stored vehicles as an object of type **`java.util.List`**
- **`getVehicle(int id)`** receives an integer id as argument and returns the corresponding vehicle. If the vehicle does not exist, the method returns **`null`**.
- **`saveVehicle(Vehicle v)`** persists the vehicle passed as argument. The method has to ensure that the id of the vehicle to be saved is not already used. If the id is already used, an **`IllegalArgumentException`** with a corresponding error message has to be thrown (see Section 7).

- **`deleteVehicle(Vehicle v)`** deletes the vehicle **`v`** from the persistent storage. If the vehicle is not available, an **`IllegalArgumentException`** with a corresponding error message has to be thrown (see Section 7).

4. Class `SerializedVehicleDAO`

The class **`SerializedVehicleDAO`** implements the interface **`VehicleDAO`** to persistently store vehicle data in a file using *object serialization.* The constructor of the class shall have an argument of type **`String`** for passing the name of the file.

In case of errors, a single-line error message with prefix "**`Error during serialization:`** " or "**`Error during deserialization:`** " should be printed using **`System.err.println()`** and the program terminated by calling **`System.exit(1)`**.

5. Class `VehicleManagement`

The class **`VehicleManagement`** implements the business logic of the application. By means of *dependency injection* the class shall obtain an object of type **`VehicleDAO`** in order to manage vehicle data.

Class **`VehicleManagement`** has to provide methods with the following functionality:

- Return all data of all vehicles
- Return all data of a specific vehicle
- Add a new vehicle
- Delete a vehicle
- Determine the total number of all vehicles
- Determine the total number of all cars
- Determine the total number of all trucks
- Compute the mean price of all vehicles
- Return the id(s) of the oldest vehicle(s)

6. Class `VehicleCLI`

Implement a Java program **`VehicleCLI`**, which realizes a command line interface (CLI) as described below. Ensure that the output of your program exactly corresponds to the examples shown.

The program **`VehicleCLI`** is to be executed as follows:

```
java VehicleCLI <file> <command>
```

`<file>`: name of the file used for persistent storage. If the file does not exist, it shall be created.

`<command>: show, add, del, count, meanprice, oldest`
For each program execution only one of the specified commands can be specified.

- **`show`**
  - Print all data of all vehicles

Example:

```
java VehicleCLI <file> show
```

Output:

```
Type:       Truck
Id:         3
```

```
Brand:      MAN
Model:      TGX 6X2
Year:       2014
Base price: 56763.00
Price:      45410.40

Type:       Car
Id:         5
Brand:      Tesla
Model:      Model S
Year:       2016
Inspection: 2016
Base price: 65000.00
Price:      55250.00

Type:       Car
Id:         2
Brand:      Opel
Model:      Manta
Year:       1972
Inspection: 2013
Base price: 21000.00
Price:      17850.00

Type:       Truck
Id:         22
Brand:      Steyr
Model:      990
Year:       1972
Base price: 6900.00
Price:      5520.00
```

- Parameter 'show <id>'
  - Print all data of a vehicle

Example:

```
java VehicleCLI <file> show 2
```

Output:

```
Type:       Car
Id:         2
Brand:      Opel
Model:      Manta
Year:       1972
Inspection: 2013
Base price: 21000.00
Price:      17850.00
```

- Parameter 'add truck <id> <brand> <model> <year built> <base price>'
  - Add a truck

Example:

```
java VehicleCLI <file> add truck 3 MAN "TGX 6X2" 2014 56763
```

- Parameter 'add car <id> <brand> <model> <year built> <base price> <inspection year>'
  - Add a car

Example:

```
java VehicleCLI <file> add car 5 Tesla "Model S" 2016 65000 2016
```

- Parameter 'del <id>'
  - Delete a vehicle

Example:
```
java VehicleCLI <file> del 3
```

- Parameter 'count'
  - Determine total number of vehicles

Example:
```
java VehicleCLI <file> count
```
Output:
```
4
```

- Parameter 'count <type>'
  - Determine total number of trucks or cars

Example:
```
java VehicleCLI <file> count truck
```
Output:
```
2
```

Example:
```
java VehicleCLI <file> count car
```
Output:
```
2
```

- Parameter 'meanprice'
  - Determine mean price of all vehicles

Example:
```
java VehicleCLI <file> meanprice
```
Output:
```
31007.60
```

- Parameter 'oldest'
  - Determine oldest vehicle(s)

Example:
```
java VehicleCLI <file> oldest
```
Output:
```
Id: 2
Id: 22
```

7. Error Messages

All exceptions due to invalid user input have to be caught and the program has to be terminated with one of the following error messages:

- `"Error: Invalid parameter."`
- `"Error: Year built invalid."`
- `"Error: Base price invalid."`
- `"Error: Inspection year invalid."`
- `"Error: Vehicle already exists. (id=<id>)"`
- `"Error: Vehicle not found. (id=<id>)"`

Example:
```
java VehicleCLI <file> add car 7 VW Golf 2005 10000
```
Output:
```
Error: Invalid parameter.
```

Example:
```
java VehicleCLI <file> add car x VW Golf 2005 10000 2015
```
Output:
```
Error: Invalid parameter.
```

Example:
```
java VehicleCLI <file> add car 7 VW Golf 2025 25000 2017
```
Output:
```
Error: Year built invalid.
```

Example:
```
java VehicleCLI <file> add car 11 Tesla "Model S" 2021 75000 2026
```
Output:
```
Error: Error: Inspection year invalid.
```

Example:
```
java VehicleCLI <file> del 4711
```
Output:
```
Error: Vehicle not found. (id=4711)
```

8. Notes

- Inputs
  If a parameter contains spaces, quotation marks can be used (see example 'add').
- Outputs
  Floating point numbers have to be formatted with the decimal delimiter '.' and exactly two decimal places. For formatting floating point numbers, the provided method `Vehicle.getDecimalFormat()` can be used.

Example:
```
Double price = 12.345;
DecimalFormat df = Vehicle.getDecimalFormat();
System.out.println("Price:       " + df.format(price));
```
Output:
```
Price:       12.35
```

9. Submission

Deadline: **Wednesday, 9.11.2022 12:00**

For developing your code, the java classes/interfaces provided in `Assignment1PLC22WS` can be used. The names of classes and the interface `VehicleDAO` must not be modified. All classes and interfaces should be left in the default package.

The **program has to be submitted before the deadline on the online platform** after it has passed all checks. Further information is provided in the lectures, tutorials and on Moodle.