INF1-OP Mock Programming Exam 2019

1. Note that all questions are compulsory.
2. Remember that a file that does not compile or does not pass the basic JUnit tests provided will get no marks.
3. This is an Open Book exam. You may bring in your own material on paper or USB thumb drive.
4. CALCULATORS MAY NOT BE USED.

General Advice

Compile and test often Make sure you compile, execute and test your code as often as possible to catch errors early on.

Main functions In this exam you are not marked on any code inside `main` functions. However, it is highly recommended that you use one for each question to test your code. Please make sure that it does not have any compilation errors before submission.

Save regularly Save all your documents regularly in case they need to be restored.

Submit regularly You can submit the same file multiple times. To avoid last minute stress, you might want to consider submitting partially correct solutions early on.

1. Scaling the Price for Cars

In this task you will extend parts of an existing online car sales platform. With this online service, users can browse and buy cars with different brands and models (e.g. Volkswagen Polo, Ford Fiesta, BMW X5).

A central class in this system is the immutable class `CarItem` which encapsulates the brand, model and current price in Great British Pound (GBP) of a single car.

Your task is to implement a subclass `ScalableCarItem`, representing a special kind of `CarItem` which scales its price based on sale numbers; the more cars of this brand and model were sold, the higher its price.

From the exam template directory, please use all of the following files to answer this question (if you are using Eclipse, make sure you import ALL of them):

- CarItem.java
- ScalableCarItemBasicTest.java

Please execute the following steps for your implementation of `ScalableCarItem`:

(a) Define the class `ScalableCarItem`, extending `CarItem`. It should have two `private` instance variables:

- A `double` variable `scaledPrice` representing the price of the car after scaling by latest car sales.
- An `int` variable `sales` representing the total amount of sold cars of the same item.

[10 marks]

PLEASE TURN OVER

(b) Write a `public` constructor which gets two `String` and one `double` parameter `brand`, `model` and `price` in that order.

Pass those three arguments on to the correct `CarItem` constructor and initialise `sales` to zero and `scaledPrice` to the value of `CarItem`'s `price` member.

[10 marks]

(c) Override `CarItem`'s `price` getter. The new implementation should return `scaledPrice` instead of `price`.

[5 marks]

(d) Implement a new `public` instance method `updateSales` which gets a single `int` parameter and returns nothing. The integer parameter is the latest number of sold cars for this item.

Check if the integer argument is larger or equal to zero. If not, throw an `IllegalArgumentException` with an error message.

This method should increment the current value of `sales` by the method's argument and then use the updated value to calculate a new `scaledPrice`.

The scaled price is the original starting price plus one percent of the original starting price for each sold item.

This method should calculate the new scaled price and save it in the `scaledPrice` member rounded to two decimal places.

For example:

```
price: 2000.50 GBP, sales = 0 --> scaledPrice = 2000.50 GBP
price: 2000.50 GBP, sales = 10 --> scaledPrice = 2200.55 GBP
price: 2000.50 GBP, sales = 25 --> scaledPrice = 2500.63 GBP
```

[20 marks]

(e) Override the method `toString` to add a representation of the subclass' member variables `sales` and `scaledPrice`.

First call `CarItem`'s version of `toString` and add a line for `sales` and `scaledPrice` using the following format:

```
Volkswagen Polo - 2042.32 GBP
sales: 7
scaled price: 2185.28 GBP
```

[15 marks]

The file you must submit for this question is `ScalableCarItem.java`. Before you submit, check that it compiles and passes the basic JUnit tests provided, otherwise it will get 0 marks.

2. Analysing Car Sales

In this question you will implement a function for the sales analysis of the online service. This module calculates various statistics for recent car sales which are provided in data files. Your task is to count the amount of cars sold for each brand based on provided data.

Car Data Sample data is provided in a data file *cars01.txt*. You can use this file to test your code or create your own version. Each line in a data file represents a single car item.

CarItem Class You will use the `CarItem` superclass from the previous question.

CarSalesAnalysis Skeleton For this question you are provided with a `CarSalesAnalysis` skeleton implementation. The skeleton has method stubs for your implementation where you can fill in your solutions.

Utils Class A utility class `CarSalesUtils` provides static methods for reading car data from a file and printing the content of collection data types to the command line.

CarSalesAnalysis main method The skeleton also comes with a main function which is already filled with an example execution of the analysis system. You are not marked on the contents of the main function, so feel free to alter it for testing your code. However, you must make sure that your final solution has *no compiler errors!* You can execute the main method the same way as you are used to from labs and tutorials. The required command line argument is the path to a car data file, e.g. `cars01.txt`.

JDK Library Classes In this question you will use the collection classes `ArrayList` and `Hashtable` which are familiar from lectures, labs and tutorials. You may need to look at the JDK documentation for it. Note that the types of your methods will involve `Map` and `List`, thus hiding, from clients, which implementation of the `Map` or `List` interface is used; your code is expected to use, concretely, `Hashtable`s and `ArrayList`s.

Argument Assumptions You may assume that none of the object arguments to your methods are `null`.

From the exam template directory, please use all of the following files to answer this question (if you are using Eclipse, make sure you import ALL of them):

- CarSalesAnalysis.java
- CarSalesUtils.java
- CarItem.java
- CarSalesAnalysisBasicTest.java
- cars01.txt

PLEASE TURN OVER

For your solution, implement the following method:

(a) Implement the method `countByBrand` in the `CarSalesAnalysis` skeleton file. This method takes a list of car items. It goes through the list and counts how many cars for each brand appear.

Corresponding counts are saved in a `Hashtable` which maps car brands to integer counts. This `Hashtable` is then returned by the method.

You can assume that none of the items in this list are `null` but the list might be empty. In the latter case, the method is expected to return an empty `Hashtable`.

The final order of car brands in the `Hashtable`'s keyset is irrelevant.

Consider the following example:

```
List parameter of car items:
Toyota Corolla - 4522.10 GBP
Chevrolet HHR - 1515.01 GBP
Toyota Celica - 5105.10 GBP
Toyota Corolla - 4600.20 GBP
Chevrolet Corvette - 8444.13 GBP
Volkswagen Polo - 6004.54 GBP

Resulting associations in returned Hashtable:
Volkswagen  -> 1
Toyota      -> 3
Chevrolet   -> 2
```

[40 marks]

The file you must submit for this question is `CarSalesAnalysis.java`. Before you submit, check that it compiles and passes the basic JUnit tests provided, otherwise it will get 0 marks.