Basic Programming

Prof. Dr. Guy Baele
1pm, January 13th, 2020, VHI 02.09/02.23/02.24

Instructions

- There are two questions. Both questions need to be answered! Each question counts for 50% of the total score for the written part of the exam.

- To hand in, create a .zip file of all your Java files named "Surname_Firstname.zip" (e.g. "Baele_Guy.zip"). Make 1 package per question and put both packages in the same .zip file. Do not submit .jar files! Only include .java files, input files we have provided should not be included. There is no need to write a report.

- The programming exam lasts for two hours at the most (the R part starts after you hand in). The Toledo system will close automatically after two hours; you can submit as many times as you want, but we will only correct your final submission. Only submissions through Toledo are accepted! Late submissions **will not be graded**!

- If you have permission to work longer on your exam, submit your solution via e-mail to both: guy.baele@kuleuven.be and mandev.gill@kuleuven.be (and check with one of us if we have received it before you leave). If you don't have permission to work longer but submit via e-mail, your e-mail **will be discarded**.

- Be sure to use the provided files for the questions from this document, which will be made available on Toledo during the length of the exam. (i.e. don't manually type out the all of the provided code in this document!). Pay attention to encapsulation!

- Don't forget to sign your name on the sheet in front before you leave.

Question 1: Object-oriented Programming

In this question, the goal is to write the required classes to make the code below run in its entirety and without error, as well as to produce the provided output below. You may not alter the provided code.

A Car is described by its brand, the specific model name, the original sale price and the mileage of the car. As such, you are required to provide a constructor that takes these arguments in this specific order (see the code comments for more details). A Truck is described by its brand, the specific model name and the original sale price.

A Vendor keeps a number of cars in stock and can sell these to a Customer, provided the latter has sufficient budget. Trucks sell at a 10% profit margin on top of the original sale price. Cars, on the other hand, are sold at a discount which depends on their mileage. If the mileage is lower than 50.000, a 10% discount on the original sale price is used; if the mileage is between 50.000 and 100.000, a 30% discount is used; for mileage over 100.000, a 50% discount applies.

Provided main class:

```java
public class SalesTest {

    public static void main(String[] args) {

        //Tesla car, Model S, with original sale price of 45.000 and mileage 20.000
        Vehicle teslaModelS = new Car(VehicleBrand.Tesla, "Model S", 45000, 20000);
        //Tesla car, Model S, with original salce price of 45.000 and mileage 65.000
        Vehicle anotherTeslaModelS = new Car(VehicleBrand.Tesla, "Model S", 45000, 65000);
        //Tesla truck, with model simply Truck, and originally priced at 150.000
        Vehicle teslaTruck = new Truck(VehicleBrand.Tesla, "Truck", 150000);

        //create first vendor - with name "Tesla Exports Ltd." - and its inventory
        Vendor teslaVendor = new Vendor("Tesla Exports Ltd.");
        teslaVendor.addVehicleToStock(teslaModelS);
        teslaVendor.addVehicleToStock(anotherTeslaModelS);
        teslaVendor.addVehicleToStock(teslaTruck);

        //Opel car, model Vectra, with original sale price of 22.875 and mileage 33.000
        Car opelVectra = new Car(VehicleBrand.Opel, "Vectra", 22875, 33000);
        //Opel truck, model Blitz, with original sale price 40.000
        Truck opelBlitz = new Truck(VehicleBrand.Opel, "Blitz", 40000);

        //create second vendor - with name "Business on the side" - and its inventory
        Vendor smallShop = new Vendor("Business on the side");
        smallShop.addVehicleToStock(opelVectra);
        smallShop.addVehicleToStock(opelBlitz);

        //create first customer - nicknamed "The collector" - with budget of 100.000
        Customer firstCustomer = new Customer("The collector", 100000);
        System.out.println(firstCustomer);
        //first customer purchases a car from the first vendor
        firstCustomer.purchaseVehicle(teslaVendor, teslaModelS);
        System.out.println(firstCustomer);

        try {
            firstCustomer.purchaseVehicle(teslaVendor, teslaModelS);
        } catch(RuntimeException rte) {
            System.err.println(rte.getMessage());
        }
        System.out.println(firstCustomer);

        firstCustomer.purchaseVehicle(teslaVendor, anotherTeslaModelS);
        System.out.println(firstCustomer);

        //create second customer - nicknamed "New in town" - with no initial budget
        Customer secondCustomer = new Customer("New in town");
        //add 5.000 to the budget of the second customer
        secondCustomer.addToBudget(5000);
        System.out.println(secondCustomer);
        try {
            secondCustomer.purchaseVehicle(smallShop, opelBlitz);
        } catch(RuntimeException rte) {
            System.err.println(rte.getMessage());
        }
        try {
            secondCustomer.purchaseVehicle(smallShop, teslaModelS);
        } catch(RuntimeException rte) {
            System.err.println(rte.getMessage());
        }

        System.out.println();
        System.out.println(VehicleBrand.Tesla + " cars owned by customer '" + firstCustomer.getName() + "': ");
        System.out.println(firstCustomer.getAllVehiclesAsString(VehicleBrand.Tesla));
        System.out.println(VehicleBrand.Opel + " cars owned by customer '" + secondCustomer.getName() + "': ");
        System.out.println(secondCustomer.getAllVehiclesAsString(VehicleBrand.Opel));

    }
}
```

<u>Required output from running the provided main class:</u>

```
Customer 'The collector' owns the following vehicles:
Cars:
Trucks:
Remaining budget: 100000

Customer 'The collector' owns the following vehicles:
Cars:
- Tesla Model S (mileage: 20000)
Trucks:
Remaining budget: 59500
Vendor 'Tesla Exports Ltd.' does not have a(n) Tesla Model S (mileage: 20000) in stock.

Customer 'The collector' owns the following vehicles:
Cars:
- Tesla Model S (mileage: 20000)
Trucks:
Remaining budget: 59500

Customer 'The collector' owns the following vehicles:
Cars:
- Tesla Model S (mileage: 20000)
- Tesla Model S (mileage: 65000)
Trucks:
Remaining budget: 28000

Customer 'New in town' owns the following vehicles:
Cars:
Trucks:
Remaining budget: 5000
Customer 'New in town' does not have sufficient budget to purchase Opel Blitz.
Vendor 'Business on the side' does not have a(n) Tesla Model S (mileage: 20000) in stock.

Tesla cars owned by customer 'The collector':
-Tesla Model S (mileage: 20000)
-Tesla Model S (mileage: 65000)

Opel cars owned by customer 'New in town':
```

Question 2: Input / Output

The file "stocktransactions.txt" is provided and may not be altered. This file describes a number of transactions for publicly traded shares/stocks on the stock market. Companies on the stock market are identified using their ticker symbol. For example, for the Intel Corporation the ticker is INTC, and for Advanced Micro Devices Inc the ticker is AMD. The input contains both company description lines and transaction lines (for more information, see below).

You can assume that all the company descriptions are at the top of the input file, and that all remaining lines are the transactions, in no pre-specified order. You don't have to anticipate other lines of any kind being inserted into the input file (including empty lines). Note that your code needs to work with larger files, i.e. with more companies and more lines in total as well.

Each company description line looks as follows:
#CSCO Cisco_Systems_Inc. Cisco Systems Inc. is an American multinational technology conglomerate headquartered in San Jose, California, …

Hence, the first field in each line is the company's ticker symbol preceded by a "#" which indicates that this line is a descriptive line. The second field – where the spaces have been replaced by underscores – is the full company name. The remaining part of the line contains a short description of the company.

Each transaction line looks as follows:
>INTC 59.15 Sell 80
In others words, there are always 4 fields on each line, separated by a space. The first field is the ticker symbol preceded by a ">" indicating that this line is a transaction line. The second field is the price per share at which a transaction occurs. The third field indicates whether shares are being bought (Buy) or sold (Sell), and the fourth field indicates how many shares are being bought or sold.

The exercise goals are listed here:

- Create a class Ticker which contains 3 fields and has the appropriate constructor(s) to create objects of the class Ticker. Implement a custom toString( ) method for this class.
- Create a class StockPosition that can store the information from a transaction line and that has the following constructor:
  StockPosition(**Ticker** ticker, **boolean** buy, **int** number, **double** price)
  Also for this class, implement a custom toString( ) method.

Use these classes (and others you deem necessary to implement) to read in the "stocktransactions.txt" file provided. The output of your program is one output file per company – the file name is the ticker symbol + ".txt" - that contains:

- A first line with the company ticker symbol
- A second line with the full company name, where the underscores from the input file are replaced with spaces.
- A third line with the company description.
- A fourth line with the final share price (i.e. of the final transaction) for that company.
- A fifth line with the average share price (up to 2 decimal spaces) across all transactions and the total number of shares across all transactions.
- The remaining lines are all the transactions for that company.

Important: the first character of each line (i.e. # or >) should not be stored nor written to the output files.

For example, here you have a simple input file with only 1 description line and 2 transaction lines for 1 single company:

```
#AMD Advanced_Micro_Devices_Inc. Advanced Micro Devices, Inc. (AMD) is an American multinational semiconductor company based in Santa Clara …
>AMD 48.38 Buy 200
>AMD 38.39 Buy 400
```

The output file should have file name AMD.txt and should look like this:

```
AMD
Advanced Micro Devices Inc.
Advanced Micro Devices, Inc. (AMD) is an American multinational semiconductor company based in Santa Clara …
38.39
43.39
600
>AMD 48.38 Buy 200
>AMD 38.39 Buy 400
```

Important: focus on reading in the file “stocktransactions.txt” only once and storing the information until you need to write the output files.

Hint: Java’s Collections Framework provides a number of general-purpose data structures, which generally can be described as “better alternatives for arrays.”