Programming Assignment 2

Submission Date : 04.08.2021
Due Date : 11.08.2021 (23:59)
Programming Languages: Java
Subject : Encapsulation, Inheritance

1 Introduction

The aim of this experiment is to learn how to use inheritance and encapsulation concepts in Object Oriented Programming (OOP). For this purpose, you are expected to write a Second Hand Car Dealer program that sells vehicles such as car, truck and bus.

2 Problem

Imagine that there is a second hand car dealer who sells various types of vehicles. These vehicles consist of petrol and electric cars, trucks and buses which are of a specific model and an age depending on the manufacture year. The car dealer sets the prices of the vehicles depending on the model and the age of the vehicle. The vehicles have different properties depending on their type:

1. The bus and truck have a tax rate.
2. The truck has a tonnage.
3. The bus has a capacity.
4. The cars have a number of doors.
5. The petrol car has a fuel capacity.
6. The electric car has a battery capacity.

You will practice inheritance and encapsulation in object oriented programming by using the common features and differences between different types of classes. Hence, analyze the input/output files and design your program accordingly. Please keep in mind that your software will be used for the checkout system of the second hand car dealer.

The second hand car dealer will be able to do few operations via your system. Please examine these operations below carefully, which will describe the requirements of your program. So you are supposed to fulfill these requirements in your program.

1. **Depositing the money:** This procedure saves the amount of money received when a vehicle is sold.
2. **Clearing the deposit:** This procedure clears the amount of money that is deposited after selling a number of vehicles and resets the records of the vehicles that are sold so far. Therefore, the system goes back to the initial state.

3. **Query the number of vehicles that are sold:** This procedure informs the user about the number of vehicles that are sold so far.
4. **Query the total amount of money that is deposited:** This procedure responds to the query regarding the amount of money that is deposited so far after selling a number of vehicles.
5. **Getting the receipt for the vehicles that are sold:** Once the user requests for a receipt for the vehicles that are sold, all information regarding the vehicles that are sold must be written to the output file.

2.1 CarDealer Class

A class called CarDealer will be provided for you in order to give information about the car dealer, such as the capacity of the dealer etc. For this purpose, CarDealer class will contain only the constants such as the maximum number of vehicles that can be sold by the second hand car dealer, the minimum and maximum age of the vehicle which is waiting to be sold, the name of the store, the maximum size of the vehicle name in letters, minimum price of the vehicle. Your code should use these constants wherever necessary!

The CarDealer class also contains the round method which takes a double number of price and returns it as a String format with two decimal points. For example, 200.755 liras would be returned as "200.76" or 90.5 liras would be returned as "90.50. You have to implement the body of the round method according to the given method signature.

An example CarDealer class is given below:

```
public class CarDealer {
	public static final int MAX_VEHICLE_AGE = 20;
	public static final int MIN_VEHICLE_AGE = 0;
	public final static int MAX_NUMBER_OF_VEHICLE_THAT_CAN_BE_SOLD = 10;
	public final static int MAX_VEHICLE_NAME_SIZE = 15;
	public final static int MIN_VEHICLE_PRICE = 0;
	public final static int MAX_VEHICLE_PRICE = 1000000;
	public static final int MIN_TRUCK_TONNAGE = 0;
	public static final int MAX_TRUCK_TONNAGE = 5000;
	public static final int MIN_BUS_CAPACITY = 0;
	public static final int MAX_BUS_CAPACITY = 100;
	public static final int MIN_BATTERY_CAPACITY = 0;
	public static final int MAX_BATTERY_CAPACITY = 600;
	public static final int MIN_FUEL_CAPACITY = 0;
	public static final int MAX_FUEL_CAPACITY = 80;

	public static String round(double value) {
			...
	}
}
```

Any initialization process that does not match with the constants given in the CarDealer class must be handled by you. Any value which is not in the boundaries specified by the mininum and maximum values will be assigned the maximum value. For example, if a value of 35 is requested to be assigned the vehicle age according to the above CarDealer class, 20 will be assigned the vehicle age. Therefore, you are not expected to print any error messages regarding these constraints.

Although the values of constants may change, the name of the constants will be the same. Please use these constants in your code if required.

2.2 Input/Output Files

You will be provided a File Stream library for reading from the input files and writing to the output files.

Input file is provided for you to test your class implementations. The output file format must match with the given output format. When you encounter with print command in the input file, you have to print the following to the output file in appropriate format which is the same with the expected output file:

- Print total number of the vehicles that are sold.
- Print all information regarding the vehicles that are sold.
- Print total amount of money of the vehicles that are sold.
- Clear the cash register.

Input and output files must be taken by your program as an argument:

java Main input.txt output.txt

The following represents the sample input file. You can change these fields whatever you want to test thoroughly your program (Please remove comments while using the sample input file).

```
Petrol 20 Mazda 150.50056 4 100        //Type Age Model Price NumberofDoors FuelCapacity
Electric 50 Toyota 70.579345 2 200     //Type Age Model Price NumberofDoors BatteryCapacity
Petrol 7 BMW 220.605890 2 150
Bus 5 MAN 520.58012 5.6 60             //Type Age Model Price TaxRate Capacity
Truck 5 BMC 520.589 4.5 1000           //Type Age Model Price TaxRate Tonnage
print
Truck 5 BMC 520.589 4.5 1000
Electric 50 Toyota 70.579345 2 200
Bus 5 MAN 520.58012 5.6 60
print
print
```

The output of your program must match with the expected output file as follows:

```
Number of Vehicles that are sold: 5

Type : Petrol Car
Model : Mazda
Age : 20
Fuel Capacity : 100
Number of Doors : 4
Price : 150.50

Type : Electric Car
Model : Toyota
Age : 20
Battery Capacity : 200
Number of Doors : 2
Price : 70.58

Type : Petrol Car
Model : BMW
Age : 7
Fuel Capacity : 150
Number of Doors : 2
Price : 220.61

Type : Bus
Model : MAN
Age : 5
Capacity : 60
Price : 549.73

Type : Truck
Model : BMC
Age : 5
Tonnage : 1000
Price : 544.02


Total Amount of Money : 1535.43
```

```
Number of Vehicles that are sold: 3

Type : Truck
Model : BMC
Age : 5
Tonnage : 1000
Price : 544.02

Type : Electric Car
Model : Toyota
Age : 20
Battery Capacity : 200
Number of Doors : 2
Price : 70.58

Type : Bus
Model : MAN
Age : 5
Capacity : 60
Price : 549.73


Total Amount of Money : 1164.33

Number of Vehicles that are sold: 0

No Vehicle that is sold

Total Amount of Money : 0.00
```

3 Javadoc

Javadoc is the JDK tool that generates API documentation from documentation comments. You are expected to document every method and class using Javadoc. Documentation comments (doc comments) are special comments in the Java source code that are delimited by the /** ... */ delimiters. These comments are processed by the Javadoc tool to generate the API docs.

The Javadoc tool can generate output originating from four different types of "source" files. In this assignment, the source code files we are interested in are your Java classes (.java) - these contain class, field, constructor and method comments.

A format of a doc comment is as follows:

- A doc comment is written in HTML and must precede a class, field, constructor or method declaration. It is made up of two parts – a description followed by block tags. In this example, the block tags are @param, @return, and @see.

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
        try {
            return getImage(new URL(url, name));
        } catch (MalformedURLException e) {
            return null;
        }
}
```

Tag Conventions

- Order of tags should be:
  - @author (classes and interfaces only, required)
  - @version (classes and interfaces only, required)
  - @param (methods and constructors only)
  - @return (methods only)
  - @exception (@throws is a synonym added in Javadoc 1.2)
  - @see
- Required tags
  - An @param tag is "required" for every parameter, even when the description is obvious.
  - The @return tag is required for every method that returns something other than void.

4 Design Notes

- Please keep in mind that this is an object oriented programming homework. Your code design will matter!
- Some default values are given to you in this document, you will also be supplied a class file named "CarDealer.java". You can find this file via Piazza. This is a JAVA class called Constants and contains variables corresponding to some values given in this document. You have to use the variables from this file while implementing your software.

Do not hard-code these values into your code. The variables in the file are defined as public variables so you can easily use them anywhere in your project. Download this file and put it into your project before you start coding. And don't forget to use it!!!

- The files specified in the pdf are going to be given as program arguments. There will be text files (input.txt, output.txt) in your working directory. The input and output files will be given as program arguments from the command line. In order to test your program, you should follow the following steps:
  - Upload your java files to your server account (dev.cs.hacettepe.edu.tr)
  - Compile your code (javac *.java)
  - Run your program (java Main input.txt output.txt)
  - Control your output data (output.txt).
- Samples of input and output files can be found on Piazza. Examine them carefully during coding.
- Do not put all the classes into the one folder. Classes must be placed in the appropriate package. Please name the packages accordingly.
- Generate UML Class Diagram from your classes in your report.

4.1 Report

Your final report should contain these topics:

1. Cover Page
2. Software Design Notes
   - Problem: Re-define the experiment in your own words. Keep the definition short by mentioning only the important parts. DO NOT COPY PASTE FROM THIS DOCUMENT.
   - Solution:
     - Explain how you approached the problem, what was the most important part of the problem from your perspective and how it effected your design. And then add anything else that you thing will further explain your design.
     - Provide a class diagram at the solution section. Below the diagram, give an explanation for each class's role in the design. Use only 1 - 2 sentences for each class. If you have more to say, you should say it at the first part of the solution section. If your class diagram is too big to be put in the report file, submit it as a jpg file with the report.

5 Submit Format

Do not submit any file via e-mail. You should upload your files via "Online Experiment Submission System" which is at http://submit.cs.hacettepe.edu.tr.

- File hierarchy must be zipped before submitted (Not .rar, only .zip files are supported by the system)

- student id.zip

< src (Main.java, *.java) >

< report.pdf, *.jpg(optional)>.

< javadoc>.

6 Grading Policy

Task	**Point**
Compiled	10
Output	60
Javadoc	10
Report	20
Total	100

7 Important Notes

- The assignment must be original, individual work. Downloaded or modified source codes will be considered as cheating. Also the students who share their works will be punished in the same way.
- We will be using the Measure of Software Similarity (MOSS) to identify cases of possible plagiarism. Our department takes the act of plagiarism very seriously. Those caught plagiarizing (both originators and copiers) will be sanctioned.
- You can ask your questions through course's piazza group and you are supposed to be aware of everything discussed in the piazza group. General discussion of the problem is allowed, but DO NOT SHARE answers, algorithms, source codes and reports .
- Ignore the cases which are not stated in this assignment and do not ask questions on Piazza for such extreme cases.
- Don't forget to write comments of your codes when necessary.
- The names of classes', attributes' and methods' should obey to Java naming convention.
- Do not submit your project without first compiling it on dev machine.
- Save all work until the assignment is graded.
- Do not miss the deadline. Submission will be end at 11/08/2021 23:59, the system will be open until 23:59:59. The problem about submission after 23:59 will not be considered.
- There will be again 3 days extensions (each day degraded by 10 points) in this project.

- Objections about your grades can be made within specified days given by TAs. Objections made outside of the specified days will not be accepted.