CSCI 1110: Assignment 2

Due: 2:00 pm, Monday, February 27, 2023

The purpose of this assignment is to reinforce your understanding of object-based programming and to problem-solve using objects, classes, and nested loops. For this problem you will be provided with sample tests in Codio. All input is done via the console and all output is to be done to the console as well. You must submit your assignment via Codio, and you can test it by submitting your code. **Note: for this assignment you are expected to install and use an IDE (other than Codio) to develop your code. Please see How-To videos in the How-To Tutorial Section of our Brightspace page on how to install an IDE**. This assignment is divided into two problems, where each problem subsumes the previous one. Hence, if you complete Problem 2, you will have also completed Problem 1.

Problem 1: Car Rental with Refill

There is a car rental company that would like to update their system with a new feature that can check if current fuel level is sufficient for the customer to drive to the destination and keep track of the cars if they need to be refilled after several trips. We read the instructions from the standard input (console/keyboard). The program will start from the main method in the **`Main.java`** file. Each object from the class **`CarModel`** is supposed to represent a certain car model with the given name, fuel economy, and tank capacity. Each object from the class **`Car`** is an instance of a car from a given model and with a certain plate number. The cars can go on a trip, and they will consume some fuel based on the length of the trip and the fuel economy of their models. Cars can be refilled after they run out of fuel.

Input

As shown in the following examples, each line of input is a command with multiple space-separated tokens:

- If the line starts with "MODEL," then it is defining a new car model, and it will be followed by the model name (comprised of English characters or digits with no spaces in between), fuel economy (floating point value), and tank capacity (floating point value).
- If the line starts with "CAR," then it defines an instance of a car, and it will be followed by the model name and the plate number (plate number is always a positive integer smaller than 999999).
- If the line starts with "TRIP," then it will be followed by a plate number and given distance in kilometers. In this case, the car will burn some gas and the program should output whether the trip was successful (see the output sample for details).
- If the line starts with "REFILL," then it will be followed by a plate number. In this case the fuel tank of the corresponding car should be refilled to be full.
- If "FINISH" is written on a line, it signals the end of the input.
- You can assume that the maximum number of commands is 100. This can help you in creating appropriate arrays (you can also use ArrayLists to handle a case where you do not know the size of an array a priori).

Examples

Input	Output

`MODEL Camry 6.5 58`
`MODEL Civic 7.5 52`
`CAR Camry 1111`
`CAR Camry 2222`
`CAR Civic 3333`
`CAR Civic 4444`
`TRIP 1111 350`
`TRIP 2222 350`
`TRIP 3333 350`
`TRIP 4444 350`
`TRIP 1111 350`
`TRIP 2222 350`
`TRIP 3333 350`
`TRIP 4444 350`
`FINISH`
	`Trip completed successfully for #1111`
	`Trip completed successfully for #2222`
	`Trip completed successfully for #3333`
	`Trip completed successfully for #4444`
	`Trip completed successfully for #1111`
	`Trip completed successfully for #2222`
	`Not enough fuel for #3333`
	`Not enough fuel for #4444`

Here is a different example with the use of REFILL command.

Input	Output

`MODEL X5 10 68`
`CAR X5 787878`
`TRIP 787878 500`
`TRIP 787878 500`
`TRIP 787878 10`
`REFILL 787878`
`TRIP 787878 500`
`FINISH`
	`Trip completed successfully for #787878`
	`Not enough fuel for #787878`
	`Not enough fuel for #787878`
	`Trip completed successfully for #787878`

Processing

Your program should be able to keep track of the trips to see if the trip for the car is successful or not. Assume every car has a full tank of fuel at the beginning of its first trip. Fuel consumption is calculated by $\frac{distance}{100.0} \times fuel\ economy.$

Apart from keeping track of the trips, your program should also have a refill function with the following assumptions.

- Your program should accommodate the input with some leading spaces and following spaces if the inputs have some extra spaces.
- You can assume that we will not have multiple "MODEL" commands with the same model name. Also, we will not have multiple "CAR" commands with the same plate number.
- You can assume that the input is always in the correct format. Also, there are no "TRIP" or "RE-FILL" commands where the plate number is not defined before. Also, there is no "CAR" command where its model name is not defined before.
- Input starting with "REFILL" means the gas tank of the corresponding car should be refilled to be full.
- The commands are run line-by-line so the order of the input (and output) matters.
- You are allowed to create new classes, use the given classes, and modify the given classes. Your final `main()` function should be defined within the class **`Main`**.

Output

If the current fuel level for the car with plate number is enough for a trip, it will output the trip completed successfully for the car with corresponding plate number; otherwise, print out not enough fuel for the car with corresponding plate number. All output should be to the console, and each line is terminated by a newline. The output format is shown in the above examples.

Problem 2: Car Rental with Trip Comparison

Extend your program from Problem 1 to handle one more change. We build on Problem 1 and add the following commands to the set of possible commands.

Input

Based on the input of Problem 1, there is one additional function that we need to cover. If an input line starts with "LONGTRIPS," then it will be followed by a plate number and a distance in kilometers (floating point).

Processing

Your program should perform the same task as stated in Problem 1. In addition, in the case of input line starting with "LONGTRIPS," you will need to output the number of trips that were made successfully by the given car and were equal or longer than the given distance. See the example output for details.

Output

The output format is shown in the following examples.

Examples

Input	Output

`MODEL Camry 6.5 58`
`MODEL Civic 7.5 52`
`CAR Camry 1111`
`CAR Civic 4444`
`TRIP 1111 50`
`TRIP 4444 50`
`TRIP 1111 350`
`TRIP 4444 350`
`TRIP 1111 350`
`TRIP 4444 350`
`LONGTRIPS 1111 300`
`LONGTRIPS 4444 300`
`FINISH`
	`Trip completed successfully for #1111`
	`Trip completed successfully for #4444`
	`Trip completed successfully for #1111`
	`Trip completed successfully for #4444`
	`Trip completed successfully for #1111`
	`Not enough fuel for #4444`
	`#1111 made 2 trips longer than 300`
	`#4444 made 1 trips longer than 300`

What to Hand In

This assignment must be submitted in Codio via the Brightspace page.

Grading

The assignment will be graded based on three criteria:

Functionality: “Does it work according to specifications?” This is determined in an automated fashion by running your program on a number of inputs and ensuring that the outputs match the expected outputs. The score is determined based on the number of tests that your program passes. So, if your program passes t/T tests, you will receive that proportion of the marks.

Completeness of Solution: “Is the implementation complete?” This considers whether the classes and methods have been implemented as specified. This is determined by visual inspection of the code. It is possible to get a good grade on this part even if you have bugs that cause the methods to fail some of the tests.

Quality of Solution: “Is it a good solution?” This considers whether the approach and algorithm in your solution is correct. This is determined by visual inspection of the code. It is possible to get a good grade on this part even if you have bugs that cause your code to fail some of the tests.

Code Clarity: “Is it well written?” This considers whether the solution is properly formatted, well documented, and follows coding style guidelines. A single overall mark will be assigned for clarity. Please see the Java Style Guide in the Assignment section of the course in Brightspace.

If your program does not compile, it is considered non-functional and of extremely poor quality, meaning you will receive 0 for the solution.

The following grading scheme will be used:

Task	100%	80%	60%	40%	20%	0%
Functionality (20 marks)	Equal to the number of tests passed.					
Solution Quality (20 marks)	Approach and algorithm are appropriate to meet requirements for both Problem 1 and 2.	Approach and algorithm are appropriate to meet requirements for Problem 1, and some requirements for Problem 2.	Approach and algorithm are appropriate to meet requirements for Problem 1, but not Problem 2.	Approach and algorithm will meet some of the requirements of Problem 1.	Reasonable attempt made at solution for Problem 1.	No code submitted or code is not a reasonable one.
Code Clarity (10 marks)	As outlined below					

Code Clarity (out of 10)

- 2 marks for identification block at the top of code.
- 2 marks for appropriate indentation when necessary.
- 2 marks for use of blank lines to separate unrelated blocks of code.

- 2 marks for comments throughout the code as needed.
- 2 marks for consistent coding style throughout.

Hints and Things to Remember

- You may need to use 1D arrays or ArrayLists.
- You **must** create and use *CarModel* and *Car* classes and objects or marks will be docked.
- All input will be correct. You do not need to handle incorrect input, for now.
- Please install and use an IDE to develop this assignment. You will only be able to submit via Codio.