# COMPSCI 2S03, Principles of Programming
# Assignment 2, Fall 2019

Hassan Ashtiani, McMaster University

Due date: Tuesday, October 1st, 9pm

- You should write your codes in Java. The input/output of the programs are all from/to standard-input/standard-output (i.e., reading from the keyboard, and writing to the text terminal).
- DOMJudge is used to automatically test your code against a number of testcases. To submit to DOMJudge, login with your username and password on http://130.113.70.122/domjudge/team from the campus (or via VPN) and then submit your source files for each question.
- In addition to DOMJudge, submit your source codes on Avenue. Upload a single zip file named `macID_assignment2.zip` (e.g., `8743649_assignment2.zip`). The zip file should include only one folder, called `8743649_assignment2`, and this folder should include all your .java source files (and nothing else). This makes marking much faster for the TAs, so **any violations from this will be penalized**.
- Your source codes will be graded by the TAs; so follow the style conventions for coding from the class (naming, indentation, spacing, comments) and good coding practise to get full marks.
- Use Piazza for discussions and clarifications about the assignment.
- This assignment has bonus points. If you get a grade above 100, then it can compensate for your other assignments (but not for the exams).

0. [**No points - for preparation**] Take a look at the given java classes (Car, CarModel, Main). The idea of this assignment is based on the car rental company that we discussed in the class. Each object from the class CarModel is supposed to represent a certain car model with the given name, fuel economy, and size of the tank. 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 their model's fuel economy. Read the given code carefully to understand what it does, and run the main() function to see the outcome.

1. [**70 points: 30 for testcases, 30 for source code, 10 for styling**] In the given Main class, we had hard-coded a main function that created a number of cars and models and sent them to a number of trips. Now we want to make our program more flexible by reading the instructions from the standard input. Look at the input example below. This particular example corresponds to the given hard-coded main() function in the Main class. As you can see, 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 gas tank size (floating point value).
   - If the line starts with "CAR", then it is defining 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 gas tank of the corresponding car should be refilled.
- 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 (later on we will learn how to handle a case where we don't know the size of an array a priori).

Input Sample

```
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
```

Output Sample

```
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 (ignore the empty lines in the beginning of the output sample; your output should not have empty lines).

Input Sample

```
MODEL X5 10 68
CAR X5 787878
TRIP 787878 500
TRIP 787878 500
TRIP 787878 10
REFILL 787878
TRIP 787878 500
FINISH
```

Output Sample

```
Trip completed successfully for #787878
Not enough fuel for #787878
Not enough fuel for #787878
Trip completed successfully for #787878
```

Some more notes about the question.

- 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 the input is always in the correct format. Also, There are no "TRIP" or "REFILL" commands where the plate number is not defined before. Also, there is no "CAR" command where its model name is not defined before.
- 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 (so you should modify the current Main class so that it reads from the standard input).

2. [**50 points: 30 for testcases, 10 for source code, 10 for styling**]

We want to build on the first question, and add the following commands to the set of possible commands.

- If an input line starts by "LONGTRIPS", then it will be followed by a plate number and a distance in kilometers (floating point). In this case, you will need to output the number of trips that were made successfully by the given car and were longer than the given distance. See the example output for details.

Input Sample

```
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
```

Output Sample

```
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
```

As this question is a super-set of the first question, any solution for it should be automatically correct for the first question as well. So you can just upload one set of source codes on Avenue (for DOMJudge, you can upload the same codes for both of the questions as long as your code passes the tests).