Final Project

[Total Marks: 100]

Class: 2019F MAD 3004
Instructor: Mohammad Kiani

Evaluation: 35%
Due Date: Thursday, 24th October 2019, 14:00 PM

Submission:

- The final project can be completed in a team of maximum 3 members.
- Prepare a report indicating each taken step in detail for the work you will have done.
- Create a zip file named as *Firstname_CollegeID_MAD3004_FP.zip* that contains all the files (with proper naming convention) related to your final project including the report and upload it on Moodle for submission.

Project name: Rally

An organizer of motorcycle and car rallies asks your help for organizing his races.

Description:

Write your code according to the description that follows:

- You are asked to create a Swift project for a rally organizer.
- In order to test the output, you should follow the execution example.

1) Vehicle class

You should firstly implement a class *Vehicle* that allows to represent a vehicle that participates in the races.

A vehicle is characterized by:

- Its name of type String like "Ferrari" for example
- Its maximal speed of type Double
- Its weight in kg of type integer
- And the level of fuel in the tank or the charge level of type integer

The class vehicle will include:

- An initializer initializing the attributes using values passed as parameters and a default initializer initializing the name with *"Anonym",* the level of fuel with zero, the maximal speed with 130 and the weight with 1000.

- The class Vehicle must be able to represent itself by producing a String containing all the characteristics of the Vehicle except for the level of fuel, representing the following format:
  <name> -> speed max = <speedMax> km/h, weight = <weight> kg
  Where <*name*> is the name of the Vehicle, <*speedMax*> is its maximal speed and <*weight*>, its weight.

- A method name *better* returning true if the current instance has a better performance than the other vehicle

- The getter properties: gets the name, gets the maximal speed, gets the weight, gets the level of fuel

Finally, this class will include and use a tool method *performance.* This method must return an estimation of the performance of the vehicle like the ratio between the maximal speed and its weight (the lighter and the faster the vehicle is, the better is its performance because it consumes less energy)

You are asked to implement the class *Vehicle* respecting a good encapsulation.

2) Cars and motorcycles

The vehicles participating in the rally can be either cars or motorcycles. A car is characterized by additional information indicating its category ("sport car", "electric car" or "hybrid car"). A motorcycle is characterized by a Boolean indicating if it has *sidecar*. By default, a motorcycle does not have a sidecar.

- The sport car has a property that indicates if the car is turbo or not.
- The sport car adopt a protocol "*Fillable"* and implements a function *fillTank* that add [60] as a fixed amount to the fuel of the car (i.e. 60)
- The electric car adopt another protocol *"Chargeable"* and implements a function *recharge* that add [40] as a fixed amount to the fuel.
- The hybrid car adopt the two protocols and add [30] and [50] by implementing recharge and fillTank functions consecutively.

- The representation of a car in the format of String will respect the following format:

  <name> -> speed max = <speedMax> km/h, weight = <weight> kg, <turbo> car category = <category>
  Where <*name*> is the name of the vehicle, <*speedMax*> is the maximal speed, <*weight*> its weight and <*category*> its category.

- The representation of a motorcycle in the format of String will respect the following format:

  <name> -> speed max = <speedMax> km/h, weight = <weight> kg, Motorcycle, with sidecar
  If it has a sidecar or:
  <name> -> speed max = <speedMax> km/h, weight = <weight> kg, Motorcycle
  If not.
  Where <*name*> is the name of the vehicle, <*speedMax*> is the maximal speed, <*weight*> its weight

3) The class GrandPrix

you are now asked to code a class *GrandPrix* as a "heterogenous collection" of vehicles. This collection represents the set of vehicles participating in a race.

This class contains a method "check" [*check () -> Bool)].* This method must allow to verify if the vehicles have the right to race together.

The class *GrandPrix* will have:

- A method "*add"* allowing to add a vehicle in the set of participants (the insertion will be done in the end of the collection)

- For a rally of the type *GrandPrix* the cars don't have the right to race with two-wheeled; the motorcycles having a "sidecar" are not considered as vehicles with two wheels; the two-wheeled have the right to race together.

In order to test the compatibility of the vehicles, you will add to the hierarchy of Vehicle a method: *"isTwoWheeled" [isTwoWheeled() -> Bool]* returning true when the vehicle is of type two-wheeled and false in the contrary case. You will consider that a basic vehicle is not a two-wheeled.

Lambton College | In Toronto

4) The race is launched

Complete the class *GrandPrix* by adding to it a method "*run" [run(turn: Int)]* simulating the progress of the race according to the following algorithm:

- Start by testing if the vehicles have the right to race together; if not, the message "Not Grand Prix" will be displayed and the method run should terminate its execution.

- When the race takes place, for every vehicle deduce as much fuel as the "*turns";* only the vehicles that still have fuel (>0) arrive to finishing line.

- If the turn is greater than 40, you need to refill the fuel tank or recharge the battery by just calling the proper methods.

- Among the vehicles that reaches the finishing line, select the most efficient one (the one which is better than all others) and display it respecting the following format:

The winner of the Grand Prix is:
<representation>

Where <*representation*> is the representation of the winning vehicle in the format of String; if no vehicle reaches the starting line, display the message:

All the vehicles failed to finish the rally

Note: Think of redoing this exercise in a protocol oriented programming. Apply the necessary modifications.

Execution example:

```
Test part 1 :
----------------
Anonym -> speed max = 130.0 km/h, weight = 1000 kg
Ferrari -> speed max = 300.0 km/h, weight = 800 kg
Renault Clio -> speed max = 180.0 km/h, weight = 1000 kg
The first car is better than the second
```

Test part 2 :

Honda -> speed max = 200.0 km/h, weight = 250 kg, motorcycle with sidecar
Kawasaki -> speed max = 280.0 km/h, weight = 180 kg, motorcycle
Lamborghini -> speed max = 320.0 km/h, weight = 1200 kg, car category = Sport
BMW -> speed max = 190.0 km/h, weight = 2000 kg, turbo car category = Sport
BMW -> speed max = 190.0 km/h, weight = 1800 kg, car category = Electric
Tesla -> speed max = 180.0 km/h, weight = 1850 kg, car category = Electric
Peugeot -> speed max = 200.0 km/h, weight = 2000 kg, car category = Hybrid

Test part 3 :
---
true
false
true
false

Test part 4 :

First round:
The winner of the Grand Prix is: Lamborghini -> speed max = 320.0 km/h, weight = 1200 kg, car category = Sport

Second round:
All the vehicles failed to finish the rally

Third round:
Not Grand Prix

Test part 5 :

First round:
The winner of the Grand Prix is: Lamborghini -> speed max = 320.0 km/h, weight = 1200 kg, car category = Sport

Second round:
The winner of the Grand Prix is: BMW -> speed max = 190.0 km/h, weight = 1800 kg, car category = Electric

Third round:
The winner of the Grand Prix is: Peugeot -> speed max = 200.0 km/h, weight = 2000 kg, car category = Hybrid
Program ended with exit code: 0