OOP Basics Retake Exam – Grand Prix

It's racing time again! Welcome to Grand Prix de SoftUni!

Overview

You should write a software which simulates a Formula 1 race under number of commands. Different number of **`Drivers`** can participate in each race and each driver … well, drives a **`Car`**. Drivers have different attitude on the track and cars have different specifications which makes the race that thrilling!

Task 1: Structure

Drivers

All drivers have a name, total time record and a car to drive:

- **`Name`** – a **string**
- **`TotalTime`** – a **floating-point number**
- **`Car`** **-** parameter of type **`Car`**
- **`FuelConsumptionPerKm`** – a **floating-point number**
- **`Speed`** – a **floating-point number**

Driver's **`Speed`** is calculated throught the formula below. Keep in mind that Speed changes on each lap.

`Speed = "(car's Hp + tyre's degradation) / car's fuelAmount"`

AggressiveDriver

This type of drivers have **`FuelConsumptionPerKm`** equal to **2.7 liters**. Also aggressive driver's **Speed** is **multiplied by 1.3.**

EnduranceDriver

This type of drivers have **`FuelConsumptionPerKm`** equal to **1.5 liters**.

Cars

Each car should keep its horsepower (Hp), fuel amount and the **type** of tyres fit at the moment

- **Hp** – an **integer**
- **`FuelAmount`** – a **floating-point number**
- **`Tyre`** **-** parameter of type **`Tyre`**

The fuel tank's **maximum capacity** of each car is **160 liters**. Fuel amount **cannot become bigger** than the tank's maximum capacity. If you are given **more fuel than** needed you should fill up the tank to the **maxiumum** and **nothing else happens**.

If **fuel amount** drops **below 0 liters** you should **throw an exception** and the **driver cannot continue** the race.

Tyres

Every type of tyre has different hardness of the compound. It also has a degradation level, which is its lifetime:

- **`Name`** – a **string**
- **`Hardness`** – a **floating-point number**
- **`Degradation`** - a **floating-point number**

Every tyre **starts** with **100** points **degradation** and drops down **towards 0**. Upon each lap it's degradation is **reduced by** the value of the **hardness**. If a tyre's degradation drops **below 0** points the tyre blows up and the **driver cannot continue** the race. If a **tyre blows up** you should **throw an exeption**.

UltrasoftTyre

Because it's ultra-soft this type of tyre has an additional property:

- **`Grip`** – a positive **floating-point number**

The name of this tyre is always "**Ultrasoft**".

Upon **each lap**, it's **`Degradation` drops down** by its **`Hardness` summed** with its **`Grip`**. Also, the ultra-soft tyre **blows up** when tyre's **`Degradation` drops below 30** points.

HardTyre

The name of this tyre is always "**Hard**". Hard tyres have less grip and slow down the car but endure bigger distance.

Task 2: Business Logic

*Overview: Each execution of the application simulates **only one** race. In the beginning, you receive info about the track (laps / length) after which drivers are registered. The start of the race is marked by the first CompleteLaps command. The race finishes when all the laps are done by the drivers.*

The Controller Class

The business logic of the program should be concentrated around several commands. Implement a class called **`RaceTower`**, which will hold the **main functionality**, represented by these **public methods**:

RaceTower.cs

```
void SetTrackInfo(int lapsNumber, int trackLength)
{
    //TODO: Add some logic here …
}
void RegisterDriver(List<string> commandArgs)
{
    //TODO: Add some logic here …
}

void DriverBoxes(List<string> commandArgs)
{
    //TODO: Add some logic here …
}

string CompleteLaps(List<string> commandArgs)
{
    //TODO: Add some logic here …
}

string GetLeaderboard()
{
    //TODO: Add some logic here …
}

void ChangeWeather(List<string> commandArgs)
{
    //TODO: Add some logic here …
}
```

NOTE: RaceTower class methods are called from the outside so these methods **must NOT** receive the command parameter (the first argument from the input line) as part of the arguments!

Method **`SetTrackInfo()`** should initialize track's total **laps number** and **length**.

Commands

There are several commands that control the business logic of the application and you are supposed to build. They are stated below.

RegisterDriver Command

Creates a **`Driver`**, and registers it into the race. Input data may **not** be **always valid**. If you **can't create** a **`Driver`** with the data provided upon this command **just skip it**. All **successfully registered** drivers should be **saved inside** the **`Racetower`** class in **any type** data structure provided by .NET Framework (no custom structures).

Parameters

- **`type`** – a **string**, equal to either "**`Aggressive`**" or "**`Endurance`**"
- **`name`** – a **string**
- **`hp`** – an **integer**
- **`fuelAmount`** – a **floating-point number**
- **`tyreType`** – a **string**
- **`tyreHardness`** – a **floating-point number**

If the type of tyre is **`Ultrasoft`**, you will receive **1 extra parameter**:

- **`grip`** – a positive **floating-point number**

Leaderboard Command

On the first line print:

`Lap {current lap}/{total laps number}`

On the next lines, all drivers should be displayed in the order of their progress in the following format:

`{Position number} {Driver's Name} {Total time / Failure reason}`

Drivers are ordered by their **`TotalTime`** in acsending order.

CompleteLaps Command

Upon this command, **all drivers** progress the race with the **specified number** of laps. **On each lap**, each driver's **`TotalTime`** should be increased with the result of the following formula:

`"60 / (trackLength / driver's Speed)"`

After each lap, you must do the **actions below** in the **exact same order**:

1. Reduce **`FuelAmount`** by: **`"trackLength * driver's fuelConsumptionPerKm"`**.
2. Degradate tyre **according** to its **type**

If you are given **greater laps number** than the number of **laps left** in the race you should **throw an exception** with the **appropriate message** and **not increment** the **completed** laps number.

After increasing **the `TotalTime`** and decreasing driver's resources (reduce FuelAmount, degradate Tyre) you should **check for overtaking** opportunities. For more info read the **Additional Action -> Overtaking** section.

Parameters

- **`numberOfLaps`** – an **integer**

Box Command

Makes a driver to box at the current lap which **adds 20 seconds** to his **`TotalTime`** and either **changes** his **tyres** with **new ones** of the specified type **or refills** with **fuel**.

Parameters

- **`reasonToBox`** - a **string**, equal to either **`ChangeTyres`** or **`Refuel`**
- **`driversName`** - a **string** specifying which driver boxes
- **`tyreType / fuelAmount`** - a **string** specifying the type of the new tyre / a **floating-point** specifying how much fuel is refilled

If the reason is **`ChangeTyres`**, you will receive **extra parameters**:

- **`tyreHardness`**- a **floating-point** specifying the new tyres hardness (only for the **ChangeTyres** case)

If the type of tyre is **`Ultrasoft`**, you will receive **1 extra parameter**:

- **`grip`** – a positive **floating-point number**

ChangeWeather Command

Changes the current weather. In the beginning, the weather is **Sunny** by default. Input parameter will **always** be **valid**!

Parameters

- **`weather`** – a **string** equal to one of the following: "**Rainy**", "**Foggy**", "**Sunny**"

Additional Action

DNF

If a driver **stops** because of **some failure** he **doesn't progress** in the race anymore under the **`CompleteLaps`** command (his stats get frozen). Drivers that are not racing anymore **still take part** at the bottom of the **Leaderboard** in the **order** of their **failure occurrence** (the latest failed driver is at the very bottom).

- The message in case of a blown tyre should be - "**`Blown Tyre`**"
- The message in case of getting out of fuel should be - "**`Out of fuel`**"
- The message in case of a crash should be - "**`Crashed`**"

Overtaking

At **certain conditions** drivers overtake each other. Generally, if a driver is **2 seconds or less behind** another driver at the end of a lap, he **overtakes** the driver **ahead** which **reduces** his **`TotalTime`** with the same interval of 2 seconds and **increases** the drivers that **has been** ahead **`TotalTime`** again with **the same interval**. Although there are some special cases:

- **`AggressiveDriver`** on **`UltrasoftTyre`** has an overtake **interval** up to **3 seconds** to the driver ahead and **crashes** **`Foggy weather`**.
- **`EnduranceDriver`** on **`HardTyre`** has an overtake **interval** up to **3 seconds** to the driver ahead and **crashes** if **attempts** an overtake in **`Rainy weather`**.

A driver is allowed to attempt an overtake **only once** in a **lap**. An overtaken driver is **not allowed** to **fight back** for his position in the same lap.

Checking for overtaking opportunities **must happen** from the **slowest** (last) driver **to** the **fastest** (first).

Task 3: Input / Output

Input

- On the **first line,** you will receive an **integer** representing the **number of laps** in the **race**
- On the **second line**, you will receive an **integer** number representing the **length** of the **track**
- On the **next** lines, you will receive **different commands**. You should **stop reading the input** when drivers **complete all laps** in the race

Below, you can see the **format** in which **each command** will be given in the input:

- **`RegisterDriver {type} {name} {hp} {fuelAmount} {tyreType} {tyreHardness}`**
- **`RegisterDriver {type} {name} {hp} {fuelAmount} Ultrasoft {tyreHardness} {grip}`**
- **`Leaderboard`**
- **`CompleteLaps {numberOfLaps}`**
- **`Box Refuel {driversName} {fuelAmount}`**
- **`Box ChangeTyres {driversName} Hard {tyreHardness}`**
- **`Box ChangeTyres {driversName} Ultrasoft {tyreHardness} {grip}`**
- **`ChangeWeather {weather}`**

Output

Below you can see what output should be provided from the commands.

Leaderboard Command

`Lap {current lap}/{total laps number}`

On the next lines, all drivers should be displayed in the order of their progress in the following format:

`{Position number} {Driver's Name} {Total time / Failure reason}`

CompleteLaps Command

If you are given invalid number of laps print on the console:

"**`There is no time! On lap {current lap}.`**"

In case of a successful overtake you should print on the console:

"**`{Overtaking driver's name} has overtaken {Overtaken driver's name} on lap {Current lap number}.`**"

Finish

After all laps in the race are completed you should print the winner on the console in the following format:

`"{Driver's name} wins the race for {TotalTime} seconds."`

The **`TotalTime`** should be rounded to **three digits** after the **decimal** point.

Constraints

- The **`Driver's name`** will be a string which may contain any ASCII character, except **space** (' ').
- The **`names`** of all drivers will always be **unique**.

- All drivers will be registered **before** the **race begins** (there won't be any **`RegisterDriver`** command after the first **`CompleteLaps`** command).
- A driver will **never box twice** in a **lap**.
- Each race **will have** a finishing driver.
- There will be **NO invalid** input data.

Examples

Input	Output
32 3 RegisterDriver Aggressive FirstDriver 650 140 Ultrasoft 0.2 3.8 RegisterDriver Endurance SecondDriver 467 78.48 Hard 0.8 RegisterDriver Endurance ThirdDriver 160 78.48 Ultrasoft 0.4 2.7 CompleteLaps 17 Leaderboard Box Refuel SecondDriver 98.28 CompleteLaps 15	Lap 17/32 1 ThirdDriver 2896.110 2 FirstDriver 6918.938 3 SecondDriver 7209.032 SecondDriver wins the race for 9838.183 seconds.
10 5 RegisterDriver Aggressive FirstDriver 650 140 Ultrasoft 10.2 3.0 RegisterDriver Aggressive SecondDriver 650 140 Hard 3.9 RegisterDriver Endurance ThirdDriver 360 78.48 Ultrasoft 2.4 0.7 CompleteLaps 14 CompleteLaps 8 Leaderboard Box ChangeTyres ThirdDriver Hard 0.3 CompleteLaps 2	There is no time! On lap 0. FirstDriver has overtaken SecondDriver on lap 1. Lap 8/10 1 ThirdDriver 931.587 2 SecondDriver 1127.098 3 FirstDriver Blown Tyre ThirdDriver wins the race for 1752.693 seconds.
14 5 RegisterDriver Endurance FirstDriver 650 140 Hard 0.2 RegisterDriver Endurance SecondDriver 650 140 Ultrasoft 0.2 0.3 RegisterDriver Aggressive ThirdDriver 350 100 Ultrasoft 0.2 0.3 RegisterDriver Aggressive FourthDriver 450 60 Hard 1.2 ChangeWeather Rainy CompleteLaps 1 Leaderboard Box Refuel FourthDriver 168 CompleteLaps 6 Box Refuel FourthDriver 2 CompleteLaps 6 Leaderboard CompleteLaps 1	Lap 1/14 1 SecondDriver 64.286 2 ThirdDriver 70.200 3 FourthDriver 143.000 4 FirstDriver Crashed Lap 13/14 1 SecondDriver 1353.124 2 FourthDriver 2122.949 3 ThirdDriver Out of fuel 4 FirstDriver Crashed SecondDriver wins the race for 1563.054 seconds.

Task 4: Bonus

Factories

You know, that the keyword **new** is a bottleneck and we are trying to use it as less as possible. We even try to separate it in new classes. These classes are called Factories and the convention for them is **{TypeOfObject}Factory.** You need to have **two different factories**, **one for Driver and one for Tyre**. This is actually a design pattern and you can read more about it. Factory Pattern

Points

For all tasks, you can submit the same project. Every different task gives you points:

Task 1.	120 points
Task 2.	180 points
Task 3.	100 points
Task 4.	25 points