## C# OOP Exam - 15 August 2020

## Overview

In this exam your task will be to create a basic Racing game. In the game there are **Map**, **Racers** and **Cars**.

## Setup

* Upload **only the** CarRacingproject in every problem **except** **Unit Tests**
* **Do not modify the interfaces or their namespaces**
* Use **strong cohesion** and **loose coupling**
* **Use inheritance and the provided interfaces wherever possible**.
  + This includes **constructors**, **method parameters** and **return types**
* **Do not** violate your **interface** **implementations** by adding **more public methods** or **properties** in the concrete class than the interface has defined
* Make sure you have **no public fields** anywhere

## Task 1: Structure (50 points)

Evaluation logic in the methods isn't included for this task.

You are given interfaces, and you have to implement their functionality in the **correct classes**.

There are **3** types of entities in the application: **Car**, **Racer**, **Map**. There should also be **CarRepository** and **RacerRepository**.

### Car

Car is a **base class** of any **type of car** and it **should not be able to be instantiated**.

#### Data

* **Make** – string
  + If the make **is null or whitespace,** throw an **ArgumentException** with message: "Car make cannot be null or empty."
* **Model** – string
  + If the model **is null or whitespace**, throw an **ArgumentException** with message: "Car model cannot be null or empty."
* **VIN** – string
  + If the VIN is **not exactly 17 characters long**, throw an **ArgumentException** with message: "Car VIN must be exactly 17 characters long."
  + All VINs will be **unique**
* **HorsePower** – int
  + If the horse power is below 0**,** throw an **ArgumentException** with message: "Horse power cannot be below 0."
* **FuelAvailable** – double
  + If the fuel available **drops** below 0**,** just **set** **it** to 0
* **FuelConsumtpionPerRace** – double
  + If the fuel consumption per race is below 0**,** throw an **ArgumentException** with message: "Fuel consumption cannot be below 0."

#### Behavior

##### void Drive()

The **Drive()** method should reduce the fuel available by the fuel consumption per race . Also when driving **TunedCar** reduces its **horse power** by 3% every time because of **engine wear**. **Horse power** should be **rounded to the closest integer number**.

#### Constructor

A **Car** should take the following values upon initialization:

string make, string model, string VIN, int horsePower, double fuelAvailable, double fuelConsumptionPerRace

#### Child Classes

There are two types of **Car**:

##### SuperCar

Constructorshould take string make, string model, string VIN and int horsePower upon initialization.

**SuperCar** always starts with 80 liters **available fuel** and 10 liters **fuel consumption per race**.

##### TunedCar

Constructorshould take string make, string model, string VIN and int horsePower upon initialization.

**TunedCar** always starts with 65 liters **available fuel** and 7.5 liters **fuel consumption per race**.

### Racer

Racer is a **base class** for any **type of racer** and it **should not be able to be instantiated**.

#### Data

* **Username** - **string**
  + If the username **is null or whitespace,** throw an **ArgumentException** with message: "Username cannot be null or empty."
  + All usernames are unique
* **RacingBehavior** - **string**
  + If the racing behavior **is null or whitespace,** throw an **ArgumentException** with message: "Racing behavior cannot be null or empty."
* **DrivingExperience** - **int**
  + If the driving experience **is below 0 or over 100,** throw an **ArgumentException** with message: "Racer driving experience must be between 0 and 100."
* **Car** - **Car**
  + If the car is null**,** throw an **ArgumentException** with message:

"Car cannot be null or empty."

#### Behavior

##### void Race()

When the **Race()** method is being called, the **Racer**'s **car** is beign driven. Also everytime **Racer** is racing, his driving experience is being increased depending on the racer type. **ProfessionalRacer** increases his **driving experience** with 10 **everytime he races** and **StreetRacer** increases his **driving experience** with 5 **every time he races**.

##### bool IsAvailable()

Returns if the **Racer** is available for a race. **Racer** is available for a race only if his **Car** has enough **fuel available** for completing a race.

#### Constructor

A **Racer** should take the following values upon initialization:

string username, string racingBehavior, int drivingExperience, ICar car

#### Child Classes

There are two types of **Racer**:

##### ProfessionalRacer

Constructorshould take the following values upon initialization:

string username, ICar car

**ProfessionalRacer** always starts with 30 **driving experience** and always have "strict" **racing behavior**.

##### StreetRacer

Constructorshould take the following values upon initialization:

string username, ICar car

**StreetRacer** always starts with 10 **driving experience** and always have "aggressive" **racing behavior**.

### Map

#### Behavior

##### string StartRace(IRacer racerOne, IRacer racerTwo)

This method calls the two players for a race. When a race is being completed, the both racers should race. If one of the racers is not available for a race, the other one automatically wins the race. If both of the racers are not available for a race method returns a message saying that the race cannot be completed. When Racer race he drives his **Car** and gains **driving experience**. Also this method should calculate which one of the racers is the **winner**. The Racer chance of winning the race depends on **his car's horse power**, **his driving experience** and **his racing behavior**. **The chance of winning a race** is calculated by **the car's horse power** multiplied by **driving experience** multiplied by **racing behavior multiplier**. If the **racing behavior** is "strict" **the multiplier is 1.2** and if the **racing behavior** is "aggressive" **the multiplier is 1.1**. All in all the chance of winning the race is:

* chanceOfWinning = horsePower \* drivingExperience \* racingBehaviorMultiplier

Return a string that says which of the racers won:

* If both of the racers are not available:
  + "Race cannot be completed because both racers are not available!"
* If one of the racers is not available:
  + "{winnerUsername} wins the race!{lostUsername}was not available to race!"
* If both racers are available and the race is completed:
  + "{racerOneUsername} has just raced against {racerTwoUsername}! {winnerUsername} is the winner!"

**Note:** There will not be a **case** where **both racers** have **the same chance of winning the race**!

### CarRepository

The **car repository** is a **repository** for all **cars** in the game.

#### Data

* Models - **a** **collection of cars (unmodifiable)**

#### Behavior

##### void Add(ICar car)

* If the car is null**,** throw an **ArgumentException** with message: "Cannot add null in Car Repository".
* **Adds** a **car** in the **collection**.

##### bool Remove(ICar car)

* **Removes** a **car** from the **collection**. **Returns true** if the removal was **sucessful**, **otherwise** - **false**.

**ICar FindBy(string property)**

* **Returns** the **car** with the **given VIN**, if there is such a **car**. **Otherwise**, returns **null**.

### RacerRepository

The **racer repository** is a **repository** for all **racers** in the game.

#### Data

* Models - **a** **collection of racers (unmodifiable)**

#### Behavior

##### void Add(IRacer racer)

* If the racer is null**,** throw an **ArgumentException** with message: "Cannot add null in Racer Repository".
* **Adds** a **racer** in the **collection**.

##### bool Remove(IRacer racer)

* **Removes** a **racer** from the **collection**. **Returns true** if the removal was **sucessful**, **otherwise** - **false**.

**IPlayer FindBy(string property)**

* **Returns** the **first** **player** with the **given username**, if there is such player. **Otherwise**, returns **null**.

## Task 2: Business Logic (150 points)

### The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces, which you have to implement in the correct classes.

**Note: The** Controller **class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!**

The first interface is **I**Controller. You must create a Controllerclass, which implements the interface and all of its methods. The constructor of Controllerdoes **not take any arguments**. The given methods should have the logic described for each in the Commands section.

### Data

You need to keep track of some things, this is why you need some private fields in your controller class:

* **cars** - **CarRepository**
* **racers** – **RacerRepository**
* **map - IMap**

### Commands

There are several **commands**, which control the **business** **logic** of the **application**. They are **stated** **below**.

#### AddCar Command

##### Parameters

* type - string
* make - string
* model - string
* VIN - string
* horsePower - int

##### Functionality

**Adds** a **Car** and **adds** it to the **CarRepository**. **Valid** types are: "**SuperCar**" and "**TunedCar**".

If the **Car** **type** is **invalid**, you have to **throw an ArgumentException** with **the following message:**

* "Invalid car type!"

If the **Car** is **added successfully**, the method should **return** the following **string**:

* "Successfully added car {carMake} {carModel} ({VIN})."

#### AddRacer Command

##### Parameters

* type - string
* username - string
* carVIN - string

##### Functionality

**Creates** a **Racer** of the **given type** and **adds** it to the **RacerRepository**. **Valid** types are: "**ProfessionalRacer**" and "**StreetRacer**".

If the **car** is **not found** throw **ArgumentException** with message:

* "Car cannot be found!"

If the racer **type** is **invalid**, throw an **ArgumentException** with message:

* "Invalid racer type!"

The **method** should **return** the following **string** if the **operation** is **successful**:

* "Successfully added racer {playerUsername}."

#### BeginRace Command

##### Parameters

* racerOneUsername - string
* racerTwoUsername - string

##### Functionality

Finds both **Racers** and they start **racing**. It returns the result from **StartRace()** method.

If one of the **racers** cannot **be found** throw **ArgumentException** with message:

* "Racer {racerUsername} cannot be found!"

#### Report Command

##### Functionality

Returns information about each racer separated with a new line. Order them by driving experience descending, then by username alphabetically. You can use the overridden **ToString() Racer** method.

"{racer type}: {racer username}"

"--Driving behavior: {racingBehavior}"

"--Driving experience: {drivingExperience}"

"--Car: {carMake} {carModel} ({carVIN})"

**Note: Use \r\n or Environment.NewLine for a new line and don't forget to trim the end if you use StringBuilder.**

#### Exit Command

##### Functionality

Ends the program.

### Input / Output

You are provided with one interface, which will help you with the correct execution process of your program. The interface is IEngine and the class implementing this interface should read the input and when the program finishes, this class should print the output.

You are given the **Engine** class with written logic in it. In order for the code to be **compiled**, some parts are **commented**, **don't forget to uncomment them**. The **try-catch block** is also **commented** in order for the program to **throw exceptions and for you to see them**, **uncomment it** when you are **ready** with this too.

#### Input

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

* **AddCar** **{type} {make} {model} {VIN} {horsePower}**
* **AddRacer {type} {username} {carVIN}**
* **BeginRace {racerOneUsername} {racerTwoUsername}**
* **Report**
* **Exit**

#### Output

Print the output from each command when issued. If an exception is thrown during any of the commands' execution, print the exception message.

#### Examples

|  |
| --- |
| **Input** |
| **AddCar SuperCar BMW M8** **JN1HJ01P0MT518872 617**  **AddCar SuperCar Audi RS6** **5N1AN08W86C526409 591**  **AddCar SuperCar Mercedes-Benz GT-R 3FAFP31382R13420 585**  **AddCar MuscleCar Ford Mustang 2FMDK36C67BB54960 670**  **AddCar TunedCar BMW 325ci-E46** **KNAFE161585017020 630**  **AddCar TunedCar Audi A4-B5 WBSDE93413CF93422** **330**  **AddCar TunedCar Toyota Supra 1C6RD6GT7CS324125 810**  **AddCar SuperCar BMW M5-Competition 4S2CK58E5S4358652 625**  **AddCar TunedCar BMW 328i-E36** **4S2CK58W0Y4300188 430**  **AddRacer ProfessionalRacer Shopoff 5N1AN08W86C526409**  **AddRacer StreetRacer Kris KNAFE161585017020**  **AddRacer ProfessionalRacer Atanas JN1HJ01P0MT518872**  **AddRacer ImaginaryRacer Pesho WBSDE93413CF93422**  **AddRacer StreetRacer Dancho 4S2CK58W0Y4300188**  **AddRacer StreetRacer Gosho 1G2ZH158664178053**  **BeginRace Kris Dancho**  **BeginRace Gosho Kris**  **BeginRace Kris Pesho**  **BeginRace Pesho Gosho**  **Report**  **Exit** |
| **Output** |
| **Successfully added car BMW M8 (JN1HJ01P0MT518872).**  **Successfully added car Audi RS6 (5N1AN08W86C526409).**  **Car VIN must be exactly 17 characters long.**  **Invalid car type.**  **Successfully added car BMW 325ci-E46 (KNAFE161585017020).**  **Successfully added car Audi A4-B5 (WBSDE93413CF93422).**  **Successfully added car Toyota Supra (1C6RD6GT7CS324125).**  **Successfully added car BMW M5-Competition (4S2CK58E5S4358652).**  **Successfully added car BMW 328i-E36 (4S2CK58W0Y4300188).**  **Successfully added racer Shopoff.**  **Successfully added racer Kris.**  **Successfully added racer Atanas.**  **Invalid racer type!**  **Successfully added racer Dancho.**  **Car cannot be found!**  **Kris has just raced against Dancho! Kris is the winner!**  **Racer Gosho cannot be found!**  **Racer Pesho cannot be found!**  **Racer Pesho cannot be found!**  **ProfessionalRacer: Atanas**  **--Driving behavior: strict**  **--Driving experience: 30**  **--Car: BMW M8 (JN1HJ01P0MT518872)**  **ProfessionalRacer: Shopoff**  **--Driving behavior: strict**  **--Driving experience: 30**  **--Car: Audi RS6 (5N1AN08W86C526409)**  **StreetRacer: Dancho**  **--Driving behavior: aggressive**  **--Driving experience: 15**  **--Car: BMW 328i-E36 (4S2CK58W0Y4300188)**  **StreetRacer: Kris**  **--Driving behavior: aggressive**  **--Driving experience: 15**  **--Car: BMW 325ci-E46 (KNAFE161585017020)** |

|  |
| --- |
| **Input** |
| **AddCar SuperCar BMW M8 JN1HJ01P0MT518872 617**  **AddCar TunedCar BMW 328i-E36 4S2CK58W0Y4300188 430**  **AddCar SuperCar Audi RS6 5N1AN08W86C526409 591**  **AddCar TunedCar BMW 325ci-E46 KNAFE161585017020 630**  **AddCar TunedCar Audi A4-B5 WBSDE93413CF93422** **330**  **AddCar TunedCar Toyota Supra 1C6RD6GT7CS324125 810**  **AddRacer ProfessionalRacer Shopoff 5N1AN08W86C526409**  **AddRacer StreetRacer Kris KNAFE161585017020**  **AddRacer ProfessionalRacer Atanas JN1HJ01P0MT518872**  **AddRacer StreetRacer Dancho 4S2CK58W0Y4300188**  **BeginRace Dancho Shopoff**  **BeginRace Dancho Atanas**  **BeginRace Dancho Kris**  **BeginRace Atanas Dancho**  **BeginRace Shopoff Atanas**  **BeginRace Shopoff Dancho**  **BeginRace Kris Atanas**  **BeginRace Kris Dancho**  **BeginRace Dancho Kris**  **BeginRace Dancho Shopoff**  **BeginRace Dancho Kris**  **BeginRace Kris Atanas**  **BeginRace Kris Shopoff**  **BeginRace Atanas Kris**  **BeginRace Shopoff Kris**  **BeginRace Kris Atanas**  **BeginRace Kris Dancho**  **Report**  **Exit** |
| **Output** |
| **Successfully added car BMW M8 (JN1HJ01P0MT518872).**  **Successfully added car BMW 328i-E36 (4S2CK58W0Y4300188).**  **Successfully added car Audi RS6 (5N1AN08W86C526409).**  **Successfully added car BMW 325ci-E46 (KNAFE161585017020).**  **Successfully added car Audi A4-B5 (WBSDE93413CF93422).**  **Successfully added car Toyota Supra (1C6RD6GT7CS324125).**  **Successfully added racer Shopoff.**  **Successfully added racer Kris.**  **Successfully added racer Atanas.**  **Successfully added racer Dancho.**  **Dancho has just raced against Shopoff! Shopoff is the winner!**  **Dancho has just raced against Atanas! Atanas is the winner!**  **Dancho has just raced against Kris! Dancho is the winner!**  **Atanas has just raced against Dancho! Atanas is the winner!**  **Shopoff has just raced against Atanas! Atanas is the winner!**  **Shopoff has just raced against Dancho! Shopoff is the winner!**  **Kris has just raced against Atanas! Atanas is the winner!**  **Kris has just raced against Dancho! Dancho is the winner!**  **Dancho has just raced against Kris! Kris is the winner!**  **Dancho has just raced against Shopoff! Shopoff is the winner!**  **Kris wins the race! Dancho was not available to race!**  **Kris has just raced against Atanas! Atanas is the winner!**  **Kris has just raced against Shopoff! Shopoff is the winner!**  **Atanas has just raced against Kris! Atanas is the winner!**  **Shopoff has just raced against Kris! Shopoff is the winner!**  **Atanas wins the race! Kris was not available to race!**  **Race cannot be completed because both racers are not available!**  **ProfessionalRacer: Atanas**  **--Driving behavior: strict**  **--Driving experience: 90**  **--Car: BMW M8 (JN1HJ01P0MT518872)**  **ProfessionalRacer: Shopoff**  **--Driving behavior: strict**  **--Driving experience: 90**  **--Car: Audi RS6 (5N1AN08W86C526409)**  **StreetRacer: Dancho**  **--Driving behavior: aggressive**  **--Driving experience: 50**  **--Car: BMW 328i-E36 (4S2CK58W0Y4300188)**  **StreetRacer: Kris**  **--Driving behavior: aggressive**  **--Driving experience: 50**  **--Car: BMW 325ci-E46 (KNAFE161585017020)** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with **Robot** and **RobotManager** classes inside. The class will have some methods, fields and one constructor, which are working properly. You are **NOT ALLOWED** to change any classes. Cover the whole class with unit tests to make sure that the class is working as intended.

You are provided with a **unit test project** in the **project skeleton**.

Do **NOT** use **Mocking** in your unit tests!