# C# OOP Regular Exam

# Formula 1

## Overview

You have to create a **Formula1** project, which stores information about pilots, cars, and races. There will be different types of cars.

## Setup

* Upload **only the** Formula1project 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.
* **Exception messages** and **output messages** can be found in the **"Utilities"** folder.
* To solve this problem use **Visual Studio 2019,** and **netcoreapp 3.1.**

## Task 1: Structure (50 points)

**For this task's evaluation logic in the methods isn't included.**

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

There are **3** types of entities in the application: **FormulaOneCar**, **Pilot**, and **Race**. There should also be **FormulaOneCarRepository**, **PilotRepository**, and **RaceRepository**.

### FormulaOneCar

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

#### Data

* **Model** - **string**
  + If the model **is null, white space,** or the length is less than **3 symbols,** throw an **ArgumentException** with the message: **"Invalid car model: { model }."**
  + All names are unique
* **Horsepower** - **int**
  + If the horsepower **is less than 900, or more than 1050,** throw an **ArgumentException** with the message: **"Invalid car horsepower: { horsepower }."**
* **EngineDisplacement** - **double**
  + If the engine displacement **is less than 1.6, or more than 2.00**, throw an **ArgumentException** with the message: **"Invalid car engine displacement: { engine displacement }."**

#### Behavior

##### double RaceScoreCalculator(int laps)

The **RaceScoreCalculator** calculates the race points in the concrete race with this formula:

**engine displacement / horsepower \* laps**

#### Constructor

The constructor of the **FormulaOneCar** class should accept the following parameters:

**string model, int horsepower, double engineDisplacement**

#### Child Classes

There are two concrete types of **FormulaOneCar**:

##### Ferrari

The constructorshould take the following values upon initialization:

**string model, int horsepower, double engineDisplacement**

**Williams**

The constructorshould take the following values upon initialization:

**string model, int horsepower, double engineDisplacement**

### Pilot

**Data**

* **FullName** - **string**
  + If the pilot**'s full name** **is null, white space** or the length is less than **5 symbols,** throw an **ArgumentException** with the message: **"Invalid pilot name: { fullName }."**
  + All names are unique
* **CanRace** - **bool**
  + Should be set to **false** as the default
* **Car** - **IFormulaOneCar**
  + If the car is null throw a **NullReferenceException** with the message: **"Pilot car can not be null."**
* **NumberOfWins** - **int**

**Behavior**

**void** **AddCar(IFormulaOneCar car)**

Sets a **car** to the **pilot**, and sets **CanRace** to true.

**void WinRace()**

The **WinRace** method increases the **NumberOfWins** by one (1) every time a pilot wins a race.

**string ToString()**

Returns a **string** with information about **the number of wins for the pilot**. The returned string must be in the following format:

**"Pilot { full name } has { number of wins } wins."**

**Constructor**

The constructor of the **Pilot** class should accept the following parameters:

### string fullName

### Race

#### Data

* **RaceName** - **string**
  + If the **race** name **is null, white space** or the length is less than **5 symbols,** throw an **ArgumentException** with the message: **"Invalid race name: { race name }."**
  + All race names are unique
* **NumberOfLaps** - **int**
  + If the number of laps **is less than 1,** throw an **ArgumentException** with the message: **"Invalid lap numbers: { number of laps }."**
* **TookPlace** - **bool**
  + Should be set to **false** as the default
* **Pilots** - **ICollection<IPilot>**

#### Behavior

##### void AddPilot(IPilot pilot)

**Adds** a **pilot** to the race.

##### string RaceInfo()

**Returns** a **string** with **information** about the **race** in the format below:

**"The { race name } race has:  
Participants: { number of participants }  
Number of laps: { number of laps }**

**Took place: { Yes/No }"**

**Note: Do not use** "**\n\r**" **for a new line.**

#### Constructor

The constructor of the **Race** class should accept the following parameters:

**string raceName, int numberOfLaps**

### FormulaOneCarRepository

The **FormulaOneCarRepository** is a **repository** for the **cars**.

#### Data

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

#### Behavior

##### void Add(IFormulaOneCar car)

* **Adds a** **formula one car** to the **collection**.

**bool Remove(IFormulaOneCar car)**

* **Removes a** formula one car from the **collection**. **Returns true** if the deletion was **successful**, **otherwise** - **false**.

**IFormulaOneCar FindByName(string model)**

* **Returns** the **first car of a given model**. **Otherwise**, returns **null**.

### PilotRepository

The **PilotRepository** is a **repository** for the **pilots**.

#### Data

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

#### Behavior

##### void Add(IPilot pilot)

* **Adds a pilot** to the **collection**.

**bool Remove(IPilot pilot)**

* **Removes** a pilot from the **collection**. **Returns true** if the deletion was **successful**, **otherwise** - **false**.

**IPilot FindByName(string fullName)**

* **Returns** the **first pilot with the given fullName**. **Otherwise**, returns **null**.

### RaceRepository

The **RaceRepository** is a **repository** for the **races**.

#### Data

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

#### Behavior

##### void Add(IRace race)

* **Adds a race** to the **collection**.

**bool Remove(IRace race)**

* **Removes** a race from the **collection**. **Returns true** if the deletion was **successful**, **otherwise** - **false**.

**IRace FindByName(string raceName)**

* **Returns** the **first race of a given model**. **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. Your task is to create a Controllerclass, which implements the interface and implements 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.

**NOTE:** When you create the Controllerclass, go into the **Engine** class constructor and uncomment the "**this.controller = new Controller();**" line.

### Data

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

* **pilotRepository** - **PilotRepository**
* **raceRepository** - **RaceRepository**
* **carRepository** - **FormulaOneCarRepository**

### Commands

There are several **commands**, which control the **business** **logic** of the **application**. They are **stated** **below**. The **Formula1 name** passed to the methods will **always** be **valid**!

#### CreatePilot Command

##### Parameters

* fullName - string

##### Functionality

**Adds** a Pilot to the **PilotRepository**.

* If a **pilot with the given full name exists**, throwan **InvalidOperationException** with **the following message:** **"Pilot { full name } is already created."**
* If the **Pilot** is **added successfully to the repository**, **return** the following **message**: **"Pilot { full name } is created."**

#### CreateCar Command

##### Parameters

* **type** - **string**
* **model** - **string**
* **horsepower** - **int**
* **engineDisplacement** - **double**

##### Functionality

Creates a **formula one car** with the given parameters and **adds** it to the **FormulaOneCarRepository**. **Valid** types are: "**Ferrari**" and "**Williams**":

* If a car with the given **model** exists, throw an **InvalidOperationException** with the message: **"Formula one car { model } is already created."**
* If the car **type** is **invalid**, throw an **InvalidOperationException** with the message: **"Formula one car type { type } is not valid."**
* If **no errors** are **thrown**, **return** a string with the following **message**: **"Car { type }, model { model } is created."**

#### CreateRace Command

##### Parameters

* raceName - string
* numberOfLaps - int

##### Functionality

**Creates** a race with the given **name**, **number** **of laps** and adds it to the **RaceRepository**:

* If a race with the given **race name exists**, **throw an InvalidOperationException** with **the following message**: **"Race { race name } is already created."**
* If **no errors** are **thrown**, **return** a string with the following **message**: **"Race { race name } is created."**

#### AddCarToPilot Command

##### Parameters

* pilotName - string
* carModel - string

##### Functionality

**Adds** a car with the given car model to a pilot with the **given name**. After successfully adding a car to a pilot, remove the car from the **FormulaOneCarRepository:**

* If the pilot **does not exist,** or thepilot already **has a car**, throwan **InvalidOperationException** with **the following message:** **"Pilot { pilot name } does not exist or has a car."**
* If the **car** **model does not exist**,throwa **NullReferenceException** with **the following message: "Car { model } does not exist."**
* If **no errors** are **thrown**, **return** a string with the following **message**: **"Pilot { pilot name } will drive a {type of car} { model } car."**

#### AddPilotToRace Command

##### Parameters

* **raceName** - **string**
* **pilotFullName** - **string**

##### Functionality

Adds a pilot with the given name, to the race with the given race name.

* If the race **does not exist**, throwa **NullReferenceException** with **the following message:** **"Race { race name } does not exist."**
* If the **pilot does not exist,** or **the pilot can not race,** or **the pilot is already in the race**,throwan **InvalidOperationException** with **the following message:** **"Can not add pilot { pilot full name } to the race."**
* If **no errors** are **thrown**, **return** a string with the following **message**: **"Pilot { pilot full name } is added to the { race name } race."**

#### StartRace Command

##### Parameters

* **raceName** - **string**

##### Functionality

If everything is valid, you should **arrange** for all pilots in the given race to start racing. As a result, this method returns **the three fastest pilots**. To execute the race you should sort all riders in **descending** order by the result of the **RaceScoreCalculator** method in **FormulaOneCar** object. In the end, if everything is valid set the race**'s TookPlace property to true, increase the winner's score, and return the corresponding message**.

* If the race **does not exist**, throwa **NullReferenceException** with **the following message: "Race { race name } does not exist."**
* If the race **has less than 3 pilots**, throwan **InvalidOperationException** with **the following message:** **"Race { race name } cannot start with less than three participants."**
* If the race **has been already executed**, throwan **InvalidOperationException** with **the following message:** **"Can not execute race { race name }."**
* If **no errors** are **thrown**, **return** a string with the following **message**:

**"Pilot { pilot full name } wins the { race name } race.**

**Pilot { pilot full name } is second in the { race name } race.**

**Pilot { pilot full name } is third in the { race name } race."**

**Note: Do not use** "**\n\r**" **for a new line.**

#### RaceReport Command

##### Functionality

Returns information about each **race** that **has been executed**. You can use the **RaceInfo** method in the **Race** class.

**"The { race name } race has:  
Participants: { number of participants }  
Number of laps: { number of laps }**

**Took place: Yes**

**The { race name } race has:  
Participants: { number of participants }  
Number of laps: { number of laps }**

**Took place: Yes**

**(…)"**

**Note: Do not use "\n\r" for a new line. There is not an empty row between different races.**

#### PilotReport Command

##### Functionality

Returns information about each pilot, ordered by the number of wins descending. You can use the override **ToString** method in the **Pilot** class.

**"Pilot {FullName} has {NumberOfWins} wins.**

**Pilot {FullName} has {NumberOfWins} wins.**

**(…)"**

**Note: Do not use** "**\n\r**" **for a new line. There is not an empty row between different reports.**

#### 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. For the code to be **compiled**, some parts are **commented on**, **don’t forget to uncomment them**.

#### Input

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

* **CreatePilot { fullName }**
* **CreateCar { type } { model } { horsepower } { engineDisplacement }**
* **CreateRace { raceName } { numberOfLaps }**
* **AddCarToPilot { pilotName } { carModel }**
* **AddPilotToRace { raceName } { pilotFullName }**
* **StartRace { raceName }**
* **RaceReport**
* **PilotReport**
* **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** |
| **CreatePilot Charles\_Leclerc**  **CreateCar Ferrari SF71H 980 1.6**  **AddCarToPilot Charles\_Leclerc SF71H**  **CreateCar Ferrari SF1000 990 1.7**  **CreatePilot Carlos\_Sainz**  **AddCarToPilot Fernando\_Alonso SF1000**  **AddCarToPilot Carlos\_Sainz SF1000**  **CreateRace Monaco\_GP 78**  **StartRace Monaco\_GP**  **CreateRace Miami\_GP 57**  **AddPilotToRace Monaco\_GP Charles\_Leclerc**  **AddPilotToRace Monaco\_GP Carlos\_Sainz**  **AddPilotToRace Monaco\_GP Fernando\_Alonso**  **CreateCar Williams FW43B 1025 1.6**  **CreatePilot Nicholas\_Latifi**  **CreateCar Pagani MCL35M 990 1.8**  **StartRace Monaco\_GP**  **AddPilotToRace Monaco\_GP Nicholas\_Latifi**  **CreatePilot Alexander\_Albon**  **CreateCar Williams FW43 1050 1.9**  **AddCarToPilot Alexander\_Albon FW43**  **AddPilotToRace Monaco\_GP Alexander\_Albon**  **AddCarToPilot Nicholas\_Latifi FW43B**  **AddPilotToRace Monaco\_GP Nicholas\_Latifi**  **StartRace Monaco\_GP**  **RaceReport**  **Exit** |
| **Output** |
| **Pilot Charles\_Leclerc is created.**  **Car Ferrari, model SF71H is created.**  **Pilot Charles\_Leclerc will drive a Ferrari SF71H car.**  **Car Ferrari, model SF1000 is created.**  **Pilot Carlos\_Sainz is created.**  **Pilot Fernando\_Alonso does not exist or has a car.**  **Pilot Carlos\_Sainz will drive a Ferrari SF1000 car.**  **Race Monaco\_GP is created.**  **Race Monaco\_GP cannot start with less than three participants.**  **Race Miami\_GP is created.**  **Pilot Charles\_Leclerc is added to the Monaco\_GP race.**  **Pilot Carlos\_Sainz is added to the Monaco\_GP race.**  **Can not add pilot Fernando\_Alonso to the race.**  **Car Williams, model FW43B is created.**  **Pilot Nicholas\_Latifi is created.**  **Formula one car type Pagani is not valid.**  **Race Monaco\_GP cannot start with less than three participants.**  **Can not add pilot Nicholas\_Latifi to the race.**  **Pilot Alexander\_Albon is created.**  **Car Williams, model FW43 is created.**  **Pilot Alexander\_Albon will drive a Williams FW43 car.**  **Pilot Alexander\_Albon is added to the Monaco\_GP race.**  **Pilot Nicholas\_Latifi will drive a Williams FW43B car.**  **Pilot Nicholas\_Latifi is added to the Monaco\_GP race.**  **Pilot Alexander\_Albon wins the Monaco\_GP race.**  **Pilot Carlos\_Sainz is second in the Monaco\_GP race.**  **Pilot Charles\_Leclerc is third in the Monaco\_GP race.**  **The Monaco\_GP race has:**  **Participants: 4**  **Number of laps: 78**  **Took place: Yes** |

|  |
| --- |
| **Input** |
| **CreatePilot Charles\_Leclerc**  **CreateCar Ferrari SF71H 980 1.6**  **AddCarToPilot Charles\_Leclerc SF71H**  **CreateCar Ferrari SF1000 990 1.7**  **CreatePilot Carlos\_Sainz**  **AddCarToPilot Carlos\_Sainz SF1000**  **CreateRace Portuguese\_GP 50**  **AddPilotToRace Portuguese\_GP Charles\_Leclerc**  **AddPilotToRace Portuguese\_GP Carlos\_Sainz**  **StartRace Spanish\_GP**  **CreatePilot Alexander\_Albon**  **CreateCar Williams FW43 1050 1.9**  **AddCarToPilot Alexander\_Albon FW43**  **CreateRace Miami\_GP 57**  **CreatePilot Nicholas\_Latifi**  **CreateCar Williams FW43B 1025 1.6**  **AddCarToPilot Nicholas\_Latifi FW43B**  **AddPilotToRace Miami\_GP Charles\_Leclerc**  **AddPilotToRace Miami\_GP Carlos\_Sainz**  **AddPilotToRace Miami\_GP Alexander\_Albon**  **AddPilotToRace Miami\_GP Nicholas\_Latifi**  **AddPilotToRace Portuguese\_GP Alexander\_Albon**  **AddPilotToRace Portuguese\_GP Nicholas\_Latifi**  **AddPilotToRace Portuguese\_GP Nicholas\_Latifi**  **CreateRace Monaco\_GP 51**  **StartRace Miami\_GP**  **PilotReport**  **StartRace Portuguese\_GP**  **RaceReport**  **Exit** |
| **Output** |
| **Pilot Charles\_Leclerc is created.**  **Car Ferrari, model SF71H is created.**  **Pilot Charles\_Leclerc will drive a Ferrari SF71H car.**  **Car Ferrari, model SF1000 is created.**  **Pilot Carlos\_Sainz is created.**  **Pilot Carlos\_Sainz will drive a Ferrari SF1000 car.**  **Race Portuguese\_GP is created.**  **Pilot Charles\_Leclerc is added to the Portuguese\_GP race.**  **Pilot Carlos\_Sainz is added to the Portuguese\_GP race.**  **Race Spanish\_GP does not exist.**  **Pilot Alexander\_Albon is created.**  **Car Williams, model FW43 is created.**  **Pilot Alexander\_Albon will drive a Williams FW43 car.**  **Race Miami\_GP is created.**  **Pilot Nicholas\_Latifi is created.**  **Car Williams, model FW43B is created.**  **Pilot Nicholas\_Latifi will drive a Williams FW43B car.**  **Pilot Charles\_Leclerc is added to the Miami\_GP race.**  **Pilot Carlos\_Sainz is added to the Miami\_GP race.**  **Pilot Alexander\_Albon is added to the Miami\_GP race.**  **Pilot Nicholas\_Latifi is added to the Miami\_GP race.**  **Pilot Alexander\_Albon is added to the Portuguese\_GP race.**  **Pilot Nicholas\_Latifi is added to the Portuguese\_GP race.**  **Can not add pilot Nicholas\_Latifi to the race.**  **Race Monaco\_GP is created.**  **Pilot Alexander\_Albon wins the Miami\_GP race.**  **Pilot Carlos\_Sainz is second in the Miami\_GP race.**  **Pilot Charles\_Leclerc is third in the Miami\_GP race.**  **Pilot Alexander\_Albon has 1 wins.**  **Pilot Charles\_Leclerc has 0 wins.**  **Pilot Carlos\_Sainz has 0 wins.**  **Pilot Nicholas\_Latifi has 0 wins.**  **Pilot Alexander\_Albon wins the Portuguese\_GP race.**  **Pilot Carlos\_Sainz is second in the Portuguese\_GP race.**  **Pilot Charles\_Leclerc is third in the Portuguese\_GP race.**  **The Portuguese\_GP race has:**  **Participants: 4**  **Number of laps: 50**  **Took place: Yes**  **The Miami\_GP race has:**  **Participants: 4**  **Number of laps: 57**  **Took place: Yes** |

## Task 3: Unit Tests (100 points)

You will receive a skeleton with **Shop** and **Smartphone** classes inside. The **Shop** class has some methods, fields, and one constructor, which are working properly. The  **Smartphone** class has three properties and a constructor. You are **NOT ALLOWED** to change any class. Cover the whole **Shop** 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!