**C# OOP Demo Exam - 07 December 2019**

* **Motocross World Championship MXGP**
* **Overview**

The FIM (Fédération Internationale de Motocyclisme) Motocross World Championship (MXGP) is the one of the biggest motocross champoinship ever. You love to ride motorcycles and you are the biggest fan on MXGP and for that reason, MXGP hired you to create platform for storing information about riders, motorcycles and races.

* **Setup**
* Upload **only the MXGP** package in every problem **except** **Unit Tests**
* **Do not modify the classes, interfaces or their packages**
* 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** in the concrete class than the interface has defined
* Make sure you have **no public fields** anywhere
* Качете само пакета MXGP във всеки проблем, с изключение на Unit Tests
* Не променяйте класовете, интерфейсите или техните пакети
* Използвайте силно сцепление и разхлабени съединители
* Използвайте наследяване и предоставените интерфейси, когато е възможно
* Това включва конструктори, параметри на метода и типове връщане
* Не нарушавайте вашите реализации на интерфейса, като добавяте повече публични методи в конкретния клас, отколкото интерфейсът е определил
* Уверете се, че нямате никъде обществени полета

Task 1: Structure (50 points)

You are given **8** interfaces, and you have to implement their functionality in the **correct classes**.

It is not required to implement your structure with **Engine**, **CommandHandler**, **ConsoleReader**, **ConsoleWriter** and enc. It's good practice but it's not required.

There are **3** types of entities and 3 repositories in the application: **Motorcycle, Rider, Race and Repository**:

Дадени са ви 8 интерфейса и трябва да внедрите тяхната функционалност в правилните класове.

Не е необходимо да прилагате вашата структура с Engine, CommandHandler, ConsoleReader, ConsoleWriter и enc. Това е добра практика, но не се изисква.

В приложението има 3 вида обекти и 3 хранилища: мотоциклет, ездач, състезание и хранилище:

Motorcycle

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

Мотоциклетът е основен клас за всеки тип мотоциклети и не трябва да може да бъде инстанциран

Data

*Model - string*

* If the model **is null, whitespace or less than 4 symbols,** throw an **ArgumentException** with message **"Model {model} cannot be less than 4 symbols."**
* All models are unique
* Ако моделът е нулев, бяло пространство или по-малко от 4 символа, хвърлете ArgumentException със съобщение "Model {model} не може да бъде по-малко от 4 символа."
* Всички модели са уникални

*HorsePower - int*

* **Every type** of motorcycle has different range of valid horsepower. If the horsepower is not in the valid range**,** throw an **ArgumentException** with message **"Invalid horse power: {horsepower}."**
* Всеки тип мотоциклети има различна гама валидни конски сили. Ако конската сила не е в допустимия диапазон, хвърлете АргументЕксцепция със съобщение "Невалидна мощност на кон: {конски сили}."

*CubicCentimeters - double*

* **Every type** of motorcycle has different cubic centimeters.
* Всеки тип мотоциклети има различни кубически сантиметри.

Behavior

*double CalculateRacePoints(int laps)*

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

CalculateRacePoints изчислява състезателните точки в конкретната надпревара с тази формула:

**cubic centimeters / horsepower \* laps**

Constructor

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

А мотоциклетът трябва да приема следните стойности при инициализация:

**string model, int horsePower, double cubicCentimeters**

Child Classes

There are several concrete types of **Motorcycles**:

Има няколко конкретни типа мотоциклети:

*PowerMotorcycle*

The **cubic centimeters** for this type of motorcycle are **450**. Minimum **horsepower** is **70** and maximum **horsepower** is **100**.

Кубичните сантиметри за този тип мотоциклети са 450. Минималните конски сили са 70, а максималните конски сили са 100.

If you receive horsepower which is not in the given range throw **ArgumentException** with message **"Invalid horse power: {horsepower}."**.

Ако получите конски сили, които не са в дадения диапазон, хвърлете ArgumentException със съобщение "Невалидна мощност на кон: {horsepower}.".

*SpeedMotorcycle*

The **cubic centimeters** for this type of motorcycle are **125**. Minimum **horsepower** is **50** and maximum **horsepower** is **69**.

Кубичните сантиметри за този тип мотоциклети са 125. Минималните конски сили са 50, а максималните конски сили - 69.

If you receive horsepower which is not in the given range throw **ArgumentException** with message **"Invalid horse power: {horsepower}."**.

Ако получите конски сили, които не са в дадения диапазон, хвърлете ArgumentException със съобщение "Невалидна мощност на кон: {horsepower}.".

Rider

Data

* **Name** - **string**
* If the name **is null, empty** or less than **5 symbols** throw an **ArgumentException** with message **"Name {name} cannot be less than 5 symbols."**
* All names are unique
* Ако името е нулево, празно или по-малко от 5 символа хвърлят ArgumentException със съобщение "Името {name} не може да бъде по-малко от 5 символа."
* Всички имена са уникални
* **Motorcycle** - **Motorcycle**
* **NumberOfWins** - **int**
* **CanParticipate - bool**
* Default behaviour is **false**
* Поведението по подразбиране е невярно
* **Rider** can participate in race, **ONLY** if he has **Motorcycle** (**Motorcycle** is not **null**)
* Ездачът може да участва в състезание, САМО ако има мотоциклет (Мотоциклетът не е нулев

Behavior

*void AddMotorcycle(Motorcycle motorcycle)*

This method adds a **Motorcycle** to the **Rider**. If the motorcycle **null**, throw **ArgumentNullException** with message **"Motorcycle cannot be null."**.

Този метод добавя мотоциклет към ездача. Ако мотоциклетът е нулев, хвърлете ArgumentNullException със съобщение "Мотоциклетът не може да бъде нулев."

If the given **Motorcycle** is not **null**, set the current **Motorcycle** as the given one and after that **Rider** can participate to race.

Ако даденият мотоциклет не е нулев, задайте текущия Мотоциклет като дадения и след това Rider може да участва в състезанието.

*void WinRace()*

When **Rider** win **Race**, number of wins should be increased.

Когато Rider спечели състезанието, броят на победи трябва да се увеличи.

Constructor

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

Rider трябва да вземе следните стойности при инициализация:

**string name**

Race

Data

*Name - string*

* If the name **is null, empty** or less than **5 symbols** throw an **ArgumentException** with message **"Name {name} cannot be less than 5 symbols."**
* All names are unique
* Ако името е недействително, празно или по-малко от 5 символа хвърлят ArgumentException със съобщение "Името {name} не може да бъде по-малко от 5 символа."
* Всички имена са уникални

*Laps - int*

* Throws **ArgumentException** with message **"Laps cannot be less than 1."**, if the laps are less than **1**.
* Хвърля ArgumentException със съобщение "обиколите не могат да бъдат по-малки от 1", ако обиколките са по-малки от 1.

Riders - A collection of Riders

Behavior

*void AddRider(Rider rider)*

This method adds a **Rider** to the **Race** if the **Rider** is valid. If the **Rider** is not valid, throw an **Exception** with the appropriate message.

Този метод добавя ездач към надпреварата, ако ездачът е валиден. Ако Rider не е валиден, хвърлете изключение със съответното съобщение.

Exceptions are:

* If a **Rider** is **null** throw an **ArgumentNullException** with message **"Rider cannot be null."**
* Ако Rider е нулев, хвърлете ArgumentNullException със съобщение "Rider не може да бъде нулев."
* If a **Rider cannot** participate in the **Race** (the **Rider** doesn't own a **Motorcycle**) throw an **ArgumentException** with message **"Rider {rider name} could not participate in race."**
* Ако един състезател не може да участва в надпреварата (ездачът не притежава мотоциклет), хвърлете ArgumentException със съобщение "Rider {rider name} не може да участва в състезанието."
* If the **Rider** already **exists** in the **Race** throw an **ArgumentNullException** with message:
* Ако Rider вече съществува в Състезанието, хвърлете ArgumentNullException със съобщение:  
  **"Rider {rider name} is already added in {race name} race."**

Repository

The repository holds information about the entity.

Data

* **Models** - A **collection of T (entity)**

Behavior

*void Add(T model)*

Adds an entity in the collection.

Добавя субект в колекцията.

*bool Remove(T model)*

Removes an entity from the collection.

Премахва образувание от колекцията.

*T GetByName(string name)*

Returns an entity with that name.

Връща субект с това име.

*Collection<T> GetAll()*

Returns all entities (unmodifiable)

Връща всички образувания (немодифицируеми)

Child Classes

Create **MotorcycleRepository**, **RiderRepository** and **RaceRepository** repositories.

Създайте хранилища на MotorcycleRepository, RiderRepository и RaceRepository.

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

**Note: The ChampionshipController class SHOULD HAVE empty constructor!**

The first interface is **IChampionshipController**. You must implement a **ChampionshipController** class, which implements the interface and implements all of its methods. The given methods should have the following logic:

Бизнес логиката на програмата трябва да се концентрира около няколко команди. Дават ви интерфейси, които трябва да внедрите в правилните класове.

Забележка: Класът ChampionshipController НЕ трябва да се справя с изключения! Тестовете са създадени да очакват изключения, а не съобщения!

Забележка: Класът ChampionshipController трябва да има празен конструктор!

Първият интерфейс е ICha ChampionshipController. Трябва да внедрите клас ChampionshipController, който реализира интерфейса и реализира всичките му методи. Дадените методи трябва да имат следната логика:

Commands

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

Има няколко команди, които контролират бизнес логиката на приложението. Те са посочени по-долу.

CreateRider Command

Parameters

* **riderName** - **string**

Functionality

Creates a **Rider** with the given name and adds it to the appropriate repository.

The method should **return** the following message:

Създава Rider с даденото име и го добавя в подходящото хранилище.

Методът трябва да върне следното съобщение:

**"Rider {name} is created."**

If a rider with the given name already exists in the rider repository, throw **an ArgumentException** with message

Ако ездач с посоченото име вече съществува в хранилището на ездача, хвърлете ArgumentException със съобщение  
**"Rider {name} is already created."**

CreateMotorcycle Command

**Parameters**

* **type - string**
* **model - string**
* **horsePower - int**

**Functionality**

Create a **Motorcycle** with the provided **model** and **horsepower** and add it to the repository. There are two types of **Motorcycle**: **"SpeedMotorcycle"** and **"PowerMotorcycle".**

Създайте Мотоциклет с предоставения модел и конски сили и го добавете в хранилището. Има два типа мотоциклети: "SpeedMotorcycle" и "PowerMotorcycle".

If the **Motorcycle** already exists in the appropriate repository throw an **ArgumentException** with following message:

Ако мотоциклетът вече съществува в подходящото хранилище, хвърлете ArgumentException със следното съобщение:

**"Motorcycle {model} is already created."**

If the **Motorcycle** is successfully created, the method should **return** the following message:

Ако мотоциклетът е успешно създаден, методът трябва да върне следното съобщение:

**"{"SpeedMotorcycle"/ "PowerMotorcycle"} {model} is created."**

AddMotorcycleToRider Command

**Parameters**

* **riderName - String**
* **motorcycleModel - String**

**Functionality**

Gives the **Motorcycle** with given name to the **Rider** with given **name** (if exists).

Дава мотоциклета с дадено име на Ездача с дадено име (ако съществува).

If the **Rider does not exist** in the **RiderRepository**, throw **InvalidOperationException** with message

Ако Rider не съществува в RiderRepository, хвърлете InvalidOperationException с съобщение

* **"Rider {name} could not be found."**

If the **Motorcycle does not exist** in the **MotorcycleRepository**, throw **InvalidOperationException** with message

Ако мотоциклетът не съществува в MotorcycleRepository, хвърлете InvalidOperationException със съобщение

* **"Motorcycle {name} could not be found."**

If everything is successful you should add the **Motorcycle** to the **Rider** and return the following message:

Ако всичко е успешно, трябва да добавите мотоциклета към ездача и да върнете следното съобщение:

* **"Rider {rider name} received motorcycle {motorcycle name}."**

AddRiderToRace Command

**Parameters**

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

**Functionality**

Adds a **Rider** to the **Race**.

If the **Race does not exist** in the **RaceRepository**, throw an **InvalidOperationException** with message:

Ако състезанието не съществува в RaceRepository, хвърлете InvalidOperationException със съобщение:

* **"Race {name} could not be found."**

If the **Rider does not exist** in the **RiderRepository**, throw an **InvalidOperationException** with message:

Ако Rider не съществува в RiderRepository, хвърлете InvalidOperationException със съобщение:

* **"Rider {name} could not be found."**

If everything is successful, you should add the **Rider** to the **Race** and return the following message:

Ако всичко е успешно, трябва да добавите ездача към състезанието и да върнете следното съобщение:

* **"Rider {rider name} added in {race name} race."**

CreateRace Command

**Parameters**

* **name - string**
* **laps - int**

**Functionality**

Creates a **Race** with the given **name** and **laps** and adds it to the **RaceRepository**.

If the **Race** with the given **name** already **exists**, throw an **InvalidOperationException** with message:

Създава състезание с даденото име и обиколки и го добавя към RaceRepository.

Ако състезанието с даденото име вече съществува, хвърлете InvalidOperationException със съобщение:

* **"Race {name} is already created."**

If everything is successful you should return the following message:

Ако всичко е успешно, трябва да върнете следното съобщение:

* **"Race {name} is created."**

StartRace Command

**Parameters**

* **raceName - string**

**Functionality**

This method is the biggest deal. If everything is valid, you should **arrange** all **Riders** and then return the three fastest **Riders**. To do this you should sort all **Rider**s in **descending** order by the result of **CalculateRacePoints** method in the **Motorcycle** object. At the end, if everything is valid **remove** this **Race** from race repository.

Този метод е най-голямата сделка. Ако всичко е валидно, трябва да подредите всички ездачи и след това да върнете трите най-бързи ездачи. За целта трябва да сортирате всички Ездачи в низходящ ред по резултат от метода CalculateRacePoints в обекта Motorcycle. В крайна сметка, ако всичко е валидно, премахнете това състезание от хранилището на състезанието.

If the **Race does not exist** in **RaceRepository**, throw an **InvalidOperationException** with message:

Ако Race не съществува в RaceRepository, хвърлете InvalidOperationException със съобщение:

* **"Race {name} could not be found."**

If the participants in the race are less than **3**, throw an **InvalidOperationException** with message:

Ако участниците в надпреварата са по-малко от 3, хвърлете InvalidOperationException със съобщение:

* **"Race {race name} cannot start with less than 3 participants."**

If everything is successful, you should return the following message:

Ако всичко е успешно, трябва да върнете следното съобщение:

* **"Rider {first rider name} wins {race name} race."  
  "Rider {second rider name} is second in {race name} race."  
  "Rider {third rider name} is third in {race name} race."**

**End Command**

**Exit** the program.

**Input / Output**

You are provided with one interface, which will help 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.

Вие имате един интерфейс, който ще помогне за правилния процес на изпълнение на вашата програма. Интерфейсът е IEngine и класът, реализиращ този интерфейс, трябва да чете входа и когато програмата завърши, този клас трябва да отпечата изхода.

**Input**

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

* **CreateRider** **{name}**
* **CreateMotorcycle** **{motorcycle type} {model} {horsepower}**
* **AddMotorcycleToRider {rider name} {motorcycle name}**
* **AddRiderToRace {race name} {rider name}**
* **CreateRace {name} {laps}**
* **StartRace {race name}**
* **End**

**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** |
| **CreateRider Michael**  **CreateRider Peter**  **CreateMotorcycle Speed Honda 60**  **CreateMotorcycle Power Suziki 80**  **CreateMotorcycle Power Yamaha 70**  **CreateRace Loket 2**  **AddMotorcycleToRider Michael Honda**  **AddMotorcycleToRider Peter Suziki**  **AddMotorcycleToRider Michael Yamaha**  **StartRace Loket**  **AddRiderToRace Loket Michael**  **AddRiderToRace Loket Peter**  **StartRace Loket**  **CreateRider Brian**  **AddRiderToRace Loket Brian**  **CreateMotorcycle Speed KTM-SX 55**  **AddMotorcycleToRider Brian KTM-SX**  **AddRiderToRace Loket Brian**  **StartRace Loket**  **End** |
| **Output** |
| **Rider Michael is created.**  **Rider Peter is created.**  **SpeedMotorcycle Honda is created.**  **PowerMotorcycle Suziki is created.**  **PowerMotorcycle Yamaha is created.**  **Race Loket is created.**  **Rider Michael received motorcycle Honda.**  **Rider Peter received motorcycle Suziki.**  **Rider Michael received motorcycle Yamaha.**  **Race Loket cannot start with less than 3 participants.**  **Rider Michael added in Loket race.**  **Rider Peter added in Loket race.**  **Race Loket cannot start with less than 3 participants.**  **Rider Brian is created.**  **Rider Brian could not participate in race.**  **SpeedMotorcycle KTM-SX is created.**  **Rider Brian received motorcycle KTM-SX.**  **Rider Brian added in Loket race.**  **Rider Michael wins Loket race.**  **Rider Peter is second in Loket race.**  **Rider Brian is third in Loket race.** |

|  |
| --- |
| **Input** |
| **CreateRider Kevin**  **CreateRider Kevin**  **CreateRider Jose**  **CreateMotorcycle Speed KTM-SX-F 100**  **CreateMotorcycle Power KTM 100**  **CreateMotorcycle Power KTM-SX-F 100**  **CreateMotorcycle Power KTM-SX-F 100**  **StartRace Imola**  **CreateRace Imola 4**  **AddRiderToRace Lommel Kevin**  **AddRiderToRace Imola Jose**  **AddRiderToRace Imola Kevin**  **AddMotorcycleToRider Kevin KTM-SX-F**  **AddRiderToRace Imola Kevin**  **CreateMotorcycle Speed Honda 60**  **CreateMotorcycle Power Suziki 80**  **CreateMotorcycle Power Yamaha 70**  **CreateRace Loket 2**  **CreateRider Michael**  **CreateRider Peter**  **AddMotorcycleToRider Michael Honda**  **AddMotorcycleToRider Peter Suziki**  **AddRiderToRace Imola Michael**  **AddRiderToRace Imola Peter**  **StartRace Imola**  **End** |
| **Output** |
| **Rider Kevin is created.**  **Rider Kevin is already created.**  **Name Jose cannot be less than 5 symbols.**  **Invalid horse power: 100.**  **Model KTM cannot be less than 4 symbols.**  **PowerMotorcycle KTM-SX-F is created.**  **Motorcycle KTM-SX-F is already created.**  **Race Imola could not be found.**  **Race Imola is created.**  **Race Lommel could not be found.**  **Rider Jose could not be found.**  **Rider Kevin could not participate in race.**  **Rider Kevin received motorcycle KTM-SX-F.**  **Rider Kevin added in Imola race.**  **SpeedMotorcycle Honda is created.**  **PowerMotorcycle Suziki is created.**  **PowerMotorcycle Yamaha is created.**  **Race Loket is created.**  **Rider Michael is created.**  **Rider Peter is created.**  **Rider Michael received motorcycle Honda.**  **Rider Peter received motorcycle Suziki.**  **Rider Michael added in Imola race.**  **Rider Peter added in Imola race.**  **Rider Peter wins Imola race.**  **Rider Kevin is second in Imola race.**  **Rider Michael is third in Imola race.** |

* **Task 3: Unit Tests (100 points)**

You will receive a skeleton with **RaceEntry, UnitMotorcycle** and **UnitRider** classes inside. The class will have some methods, properties, fields and one constructor, which are working properly. You are **NOT ALLOWED** to change any class. Cover the whole class (**RaceEntry**) 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 modify its NuGet packages**.

Note: The **TheRace** you need to test is in the **global namespace**, so **remove any using statements,** pointing towards the namespace **TheRace**.

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