Exercise: Reflection and Attributes

Problems for exercise and homework for the "C# OOP" course @ SoftUni".

You can check your solutions here: https://judge.softuni.org/Contests/1521/Reflection-and-Attributes-Exercise

I. Reflection

1. Command Pattern

Skeleton provided

Create a command pattern design using reflection. Use the provided skeleton, which contains a public **StartUp** class with a written logic inside the **Main** method. It is commented, so when you write the logic of your program, you can uncomment the code and test it. The **input of commands** will be received until the "**Exit**" **command**. Each **command line** will look as it follows: "**{CommandName} {CommandArgs}**". **CommandName** will be as follows: "**Hello**" -> executing **HelloCommand** and so on.

There are a few steps you can follow to employ the command pattern design:

Create a Command Interface

Create a Command interface - **ICommand**, which contains a method - **Execute(string[] args)**.

Create Command Objects

For this exercise, you need to create the following commands:

HelloCommand - The result from its execution should be: **$"Hello, {args[0]}".**

ExitCommand - It should **exit the program** and return **null.**

Create a Command Interpreter Interface

Create a command interpreter interface **ICommandInterpreter**, which contains a method **Read(string args)**.

Create a Command Interpreter Class

Create a class **CommandInterpreter**. Its purpose is to **read** commands, **execute** them and **return the result** from their execution.

Create a Class Engine

Create a class Engine, which contains a **void Run()** method. It should hold the following field:

```
private readonly ICommandInterpreter commandInterpreter;
```

It should have a constructor, which accepts an **ICommandInterpreter**:

```
public Engine(ICommandInterpreter commandInterpreter)
```

The **Run()** method should **accept input** from the console and pass it to the proper class, as well as **print** the output from the commands.

II. Attributes

1. Validation Attributes

Skeleton provided

Create your custom validation attributes. There is a written logic inside the provided skeleton, which you can use to test the logic of your program. It is commented, so when you write the logic of your program, you can uncomment the code and test it.

Create Attributes

Create a validation attribute: **`MyValidationAttribute`**. Its purpose is to **validate properties**.

- It should contain the following method: `public abstract bool IsValid(object obj)`

Create a validation attribute: **`MyRangeAttribute`**.

- Its constructor should accept two parameters - **`int minValue, int maxValue`**, which represent a range of integer numbers
- It should contain two fields: **`int minValue`** and **`int maxValue`**
- It should implement the `bool IsValid(object obj)` method and its logic should validate whether the passed **`object obj`** parameter is within the set range

Create a validation attribute: **`MyRequiredAttribute`**.

- It should implement the `bool IsValid(object obj)` method and its logic should validate whether a property has the attribute or not

Create an Entity

Create a class Person. It should have a constructor, which accepts two parameters: `string fullName, int age`.

It should have two properties:

- **`string FullName`** - the property is required. Apply the **`MyRequiredAttribute`**
- **`int Age`** - the age should be between 12 and 90. Apply the **`MyRangeAttribute`** and set the right values for minimum and maximum age

Create a Validator Class

Create a **`static class Validator`**. It should contain a method - `public static bool IsValid(object obj)`, which must validate the properties of a given object.