**School system - part 2**

Design Patterns Exam, 11 November 2016

Refactor and complete a school management system.

**Application description**

About a month ago you were given [the task](https://github.com/TelerikAcademy/High-Quality-Code-Part-2/blob/master/Exam/Task/Description.md) to fix the crappy system which a team of Indian developers produced. You have done some good job, so the clients, seeing the great progress, raised their expectations and asked your company to deliver a lot more than the initial agreement. After you and your team reviewed the deadlines, you realized that you cannot finish the project on your own. Your manager talked with the company executives and they agreed that the work will be split between two teams and they also decided to spend part of the budget for consultancy from a highly skilled software architect. As you probably know, these guys are paid to give advices and if you want them to produce some code - it is twice as expensive. Since the budget for this project is not endless, your bosses were able to pay the architect for just producing some document with guidelines and a tiny bit of his software magic. Following is a list with advices/action items which were produced by the architect and you need to read them carefully because there lies the key to success:

* I have split the system to 2 projects: the first one is the framework which lays the foundations of the software; the second one is the CLI/client where you will use and upgrade the framework with custom requirements from the client.
* The split is an essential part of the whole process, because this way your both teams will be able to advance in parallel. Of course this does come with a cost. The team implementing the framework should build it as a "real" framework, where a client can easily plug in and change everything, so that the end product meets the customer requirements.
* Building a Framework means that you need to leverage the full capabilities of the SOLID principles and especially the Dependency Inversion principle. What does this mean is, in order to build it successfully, you cannot afford any of your modules/types to depend on other of your modules/types, so that you can change them freely.
* On the client side of the project you will need to utilize an IoC container, so that all of the goodies of the dependency inversed framework can be assembled easily.
* In order to meet the performance specifications and to be able to monitor your system, you will need to have a testing environment where you will measure and test your software.
* Last but not least, you will need to cover the system with tests.
* What I have done for you in terms of software development is the following:
  + Extracted a new Framework project where you will build your framework's internals.
  + In your client I installed an IoC container (**Ninject**) and added a **ConfigurationManager** which will enable you to add code for measurement/testing purposes and run it only on your test environment (based on configuration in your **App.config** file).

This is everything you've got from the architect, so now it's your turn to build a successful product using these advices.

**Input**

The input data is the same as it was before. It comes from the console, consists of a sequence of commands, each staying at a separate single line. The input ends by the "End" command.

**Output**

The output is also the same and should be on the console. It should consist of the outputs from each of the commands from the input sequence. There is however a new requirement which comes from the architect and it says that if the project is run with **IsTestEnvironment** with a value of **true**, you should add some performance measurements.

**Hint 1:** Check what's going on in the following files in the CLI project (the architect left some goodies) - **IConfigurationProvider**, **ConfigurationProvider**, **SchoolSystemModule**, **App.config**.

**Hint 2:** The null tests are your friends. Let them run. This time you have two output files for each test - the first one is run with **IsTestEnvironment** with a value of **false** (and has the same output as before) and the second one is run with **IsTestEnvironment** with a value of **true** (and has some additional measurement lines).

**Application requirements**

Follow the good object orientated programming and (especially) dependency inversion practices and principles. Modifying things that you are NOT allowed to modify will result in a penalty for each violation.

**Problem 1. Code Refactoring (40 points)**

Refactor the code in both Framework and CLI projects following the best practices introduced in the Design Patterns course, so that you comply with the architect's advices:

* Replace all usages of the **new** keyword with a corresponding architectural approach (design pattern) in the following classes: **Startup**, **CreateStudentCommand**, **CreateTeacherCommand**, **Teacher** (except internal .NET classes such as **StringBuilder**, **List**, **Exception** types and etc.).
* Replace the usage of the **Activator.CreateInstance** in **CommandParserProvider** with a corresponding architectural approach (design pattern).
* Extract and encapsulate the usages of the **Teachers** and **Students** dictionaries.
* Modify the architectural approach around the **CreateStudentCommand** and **CreateTeacherCommand** classes and their **currentStudentId** and **currentTeacherId** fields - the fields should not be static, but should still represent the same logic for keeping reference to the corresponding current Id. (student/teacher factory with ISingleton)
* Use an IoC container to compose all of the refactored code.

**Problem 2. Unit Tests (30 points)**

Design and fully implement **unit tests for**:

* **CreateStudentCommand class**
* **RemoveStudentCommand class**
* **TeacherAddMarkCommand class**

Any other code is **not required** to be tested. You should cover the **Execute** methods. Be sure to test all major execution scenarios + all interesting border cases and special cases. You can use VSTT, VSTT v2 or NUnit. At least one of your tests should use mocking with JustMock or Moq.

**Problem 3. New Features (30 points)**

Implement code for performance measurement (execution time measurement) based on the configuration in the App.config file. **You are NOT allowed to change or move in other projects the IConfigurationProvider interface and ConfigurationProvider class**:

* Implement code which prints out the execution time of the following methods (which you should have implemented as part of Problem 1):
  + **GetCommand** method of **ICommandFactory** interface
  + **CreateStudent** method of **IStudentFactory** interface
  + **CreateMark** method of **IMarkFactory** interface
  + The code should run (print out measurement results) **ONLY** when **IsTestEnvironment** is set to **true**
* The output should match the output from the Null tests (those marked with **IsTestEnvironment = true**)
* The time in milliseconds (the actual number) **CAN** be different in your executions but **CANNOT** be some fixed constant in the code.