**Application description**

An elite team of Indian software developers were tasked to develop a new system for the local schools. They had six months to complete this task and at the end, they were supposed to present this system to the headmasters of all the schools. The months quickly passed and the big day came but they were not ready. They probably didn't watch this [TED Talk](https://www.ted.com/talks/tim_urban_inside_the_mind_of_a_master_procrastinator?language=en). They knew they had at least three bugs in their system, even some of the code was in Hindi, but they decided to present anyways. Guess what happened - the demo failed, their system crashed several times and the two commands for creating and removing teachers were not even implemented! It was a total disaster. The headmasters were left with no choice. The new school year had already started and they needed that system, now! But first, lets give you a quick overview of the system:

* **Students** have a first name, last name, grade and a collection of marks. Both names must have a length between 2and 31 symbols long and it can consist only of characters of the latin alphabet. Each student cannot have more than 20marks.
* **Grades** can be between 1 and 12.
* **Teachers** have a first name, last name and a subject that they teach. Both names must have a length between 2 and 31 symbols long and it can consist only of characters of the latin alphabet. They also poses the *mighty power* to add marks to students. Poor kids...
* **Marks** have a subject and a value that can be between 2 and 6.

Now that you have a basic idea of the objects that were implemented by the developers, it's time to get familiar with the user interface. The program takes as input commands as strings. Each command consists of a name and a collection of parameters. When parsing a command, the command parser looks for a class that implements the ICommand interface and contains the name of the command within the name of the class (case insensitive). If the class does not implement the ICommand interface, it won't be detected by the reflection algorithm.

* **CreateStudent *[FirstName] [LastName] [Grade]*** - Creates a new student with the provided first name, last name and grade and prints out a success message that contains the full name, grade and the ID of the created student.
* **CreateTeacher *[FirstName] [LastName] [SubjectId]*** - Creates a new teacher with the provided first name, last nameand subject and prints out a succes message that contains the full name, subject and the ID of the created teacher.
* **RemoveStudent *[StudentId]*** - Removes a student with the provided ID and prints out a success message.
* **RemoveTeacher *[TeacherId]*** - Removes a teacher with the provided ID and prints out a success message.
* **StudentListMarks *[StudentId]*** - Lists the marks of a student with the provided ID and prints out a list of marks in the format SUBJECT => MARKVALUE.
* **TeacherAddMark *[TeacherId] [StudentId] [Mark]*** - The teacher with the provided ID adds a mark to the student with the provided ID with the provided value.

In case of any error during command execution, it should print out a descriptive error message containing all the necessary information. The IDs of the created objects are **persistent**, which means that if you delete one, the others do not rearrange.

**Input**

The input data comes from the console. It consists of a sequence of commands, each staying at a separate single line. The input ends by the **"End"** command.

**Output**

The output should be on the console. It should consist of the outputs from each of the commands from the input sequence.

**Constraints**

* The command name will always be separated by the space symbol (" ") from the parameters and each parameter will be separated by the space symbol as well.
* Numbers will be valid 32-bit signed integer numbers.

**Hint 1:** Some variables and comments are in Hindi. Please forgive me. <https://translate.google.com/#hi/en/>

**Hint 2:** The null tests are your friends. Let them run.

**Hint 3:** Project > Right Click > Properties > Application

**Application requirements**

Follow the good object orientated programming and 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 (35 points)**

Refactor the source code to improve its quality following the best practices introduced in the courses High-Quality Code Part 1 and Part 2. **You are NOT allowed to change the ICommand interface. (except to add documentation)** Find common refactoring problems, including:

* Fix compilation errors and warnings
* Remove redundant usings, comments and constants
* Introduce constants, fields, properties and local variables
* Apply better naming of classes, variables, fields, properties and parameters
* Fix solution name and namespace names
* Extract classes, interfaces and enumerations into appropriate namespaces
* Remove code duplication and redundancy
* Implement property validations and guard clauses
* Remove middle man

**Problem 2. Unit Testing (25 points)**

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

* **Engine class** (12 points)
* **Teacher class** (4 points)
* **Student class** (2 points)
* **Mark class** (2 points)

Any other code is **not required** to be tested. You should cover all public members (test the constructors, properties and 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. **You are NOT allowed to modify System.Console's input or output streams. That would result in an Integration test. Extract providers.**

**Problem 3. Bug Fixing (12 points)**

Debug the code and find and fix the four bugs in it. Describe them in the provided text file Documentation.txt. If you cannot fix them, just describing them in the file awards you points.

**Problem 4. New Features (12 points)**

Add the following new features:

* **Implement the CreateTeacher command** so that it creates a Teacher with the passed parameters and saves it. It should work with the syntax described above and return appropriate success and error messages.
* **Implement the RemoveTeacher command** so that it removes the Teacher with the passed ID. It should work with the syntax described above and return appropriate success and error messages.

**Problem 5. StyleCop (6 points)**

You have been provided with a StyleCop ruleset and it's located in Docs namespace. It's configured to analyze your code each time you Build it. There is no need to install any additional versions of StyleCop. You can find all warnings in the Error List view. The StyleCop warnings have a code in the following format: SA0000, where the zeros are some numbers that you shouldn't care about. Find all StyleCop warnings and fix them. **You are NOT allowed to modify the ruleset in any way.**

**Problem 6. Code Documentation (4 points)**

Document the **interface** and **methods** of the **ICommand** interface as well as **all other interfaces you abstract**, using XML documentation. Any other code documentation is not required.

**Problem 7. Performance Bottleneck (6 points)**

Find the performance bottleneck and briefly describe it in the provided text file Documentation.txt. Fix the problem if possible. If you cannot fix it, just describing it in the file awards you points.

**Null tests**

You can also find them in the Tests folder.

**01. Input**

CreateStudent Pesho Peshev 11

CreateStudent Gosho Peshev 9

StudentListMarks 1

**01. Expected output**

A new student with name Pesho Peshev, grade Eleventh and ID 0 was created.

A new student with name Gosho Peshev, grade Ninth and ID 1 was created.

This student has no marks.

**02. Input**

CreateStudent Pesho Petrov 10

CreateTeacher Gosho Vesheff 2

TeacherAddMark 0 0 3

**02. Expected output**

A new student with name Pesho Petrov, grade Tenth and ID 0 was created.

A new teacher with name Gosho Vesheff, subject Math and ID 0 was created.

Teacher Gosho Vesheff added mark 3 to student Pesho Petrov in Math.

**03. Input**

CreateStudent Pesho Petrov 6

CreateStudent Gosho Petrov 6

CreateTeacher Gosho Vesheff 2

TeacherAddMark 0 1 3

**03. Expected output**

A new student with name Pesho Petrov, grade Sixth and ID 0 was created.

A new student with name Gosho Petrov, grade Sixth and ID 1 was created.

A new teacher with name Gosho Vesheff, subject Math and ID 0 was created.

Teacher Gosho Vesheff added mark 3 to student Gosho Petrov in Math.

**04. Input**

CreateStudent Pesho Petrov 6

CreateTeacher Gosho Vesheff 2

TeacherAddMark 0 0 3

TeacherAddMark 0 0 2

CreateTeacher Stamat Shop 1

CreateStudent Gosho Petrov 6

TeacherAddMark 1 1 5

TeacherAddMark 1 1 4

TeacherAddMark 1 0 3

StudentListMarks 0

RemoveStudent 0

StudentListMarks 0

StudentListMarks 1

RemoveTeacher 1

**04. Expected output**

A new student with name Pesho Petrov, grade Sixth and ID 0 was created.

A new teacher with name Gosho Vesheff, subject Math and ID 0 was created.

Teacher Gosho Vesheff added mark 3 to student Pesho Petrov in Math.

Teacher Gosho Vesheff added mark 2 to student Pesho Petrov in Math.

A new teacher with name Stamat Shop, subject English and ID 1 was created.

A new student with name Gosho Petrov, grade Sixth and ID 1 was created.

Teacher Stamat Shop added mark 5 to student Gosho Petrov in English.

Teacher Stamat Shop added mark 4 to student Gosho Petrov in English.

Teacher Stamat Shop added mark 3 to student Pesho Petrov in English.

The student has these marks:

Math => 3

Math => 2

English => 3

Student with ID 0 was sucessfully removed.

The given key was not present in the dictionary.

The student has these marks:

English => 5

English => 4

Teacher with ID 1 was sucessfully removed.