High-Quality Code Exam – Theatre

A junior developer Huy Phuong Nguyen from Vietnam was once assigned to design and implement an information system for theatres and performances. The original problem statement is given below:

In the city there are several theatres. Each of them has a name and a timetable that holds all performances, their start date and time, their duration and ticket price. Your task is to model the theatres, performances and the timetable and create a console-based application that executes a sequence of commands:

- **AddTheatre(theatre)** – adds a theatre by given **theatre** name. The theatre name is unique among all theatres. Prints "**Theatre added**" as result or "**Error: Duplicate theatre**" in case of name conflict.
- **PrintAllTheatres()** – prints all theatres in the city, ordered by name, separated by comma and space, or "**No theatres**" in case of empty list of theatres.
- **AddPerformance(theatre, performance, date-and-time, duration, price)** – adds a **performance** to the specified **theatre** with the specified **date-and-time**, **duration** and ticket **price**. Performances are not allowed to overlap: at most one performance can take place at given moment of time for given theatre. Prints "**Performance added**" as a result or "**Error: Theatre does not exist**" in case of non-existing theatre or "**Error: Time/duration overlap**" in case of overlapping performances.
- **PrintAllPerformances()** – prints all performances in the city in format **(performance, theatre, date-and-time)**, ordered by theatre (as first criteria) and by date and time (as second criteria), separated by comma and space, or "**No performances**" in case of empty list of performances.
- **PrintPerformances(theatre)** – prints all performances for the specified theatre in format **(performance, date-and-time)**, ordered by date and time, separated by comma and space, or "**No performances**" in case of empty list of performances for the specified theatre. Print "**Error: Theatre does not exist**" in case of non-existing theatre.

The **input** could contain up to 50 000 commands so the efficiency is important. All strings in the commands (e.g. theatre names and performance titles) consist of alphabetical characters, numbers and spaces. Prices are real numbers with up to 2 digits after the decimal point "**.**" (e.g. 122.55, or 220). Date and time is always given and printed in 24-hours format: **dd.MM.yyyy HH:mm** (e.g. 23.01.2015 09:20). Empty lines in the input should be skipped.

The **output** should be printed at the console and should contain one text line corresponding to each command from the input.

Sample Input

```
PrintAllPerformances()
PrintAllTheatres()
AddTheatre(Theatre Sofia)
AddTheatre(Theatre 199)
AddTheatre(Theatre Sofia)
PrintAllTheatres()
AddTheatre(Ivan Vazov National Theatre)
AddPerformance(Theatre 199, Duende, 20.01.2015 20:00, 1:30, 14.5)
AddPerformance(Theatre 199, Bella Donna, 20.01.2015 20:30, 1:00, 12)
AddPerformance(Theatre 199, Bella Donna, 20.01.2015 21:30, 1:00, 12)
AddPerformance(Theatre Sofia, Bella Donna, 20.01.2015 19:30, 1:00, 15)
AddPerformance(Theatre Sofia, Don Juan, 20.01.2015 20:31, 2:00, 14.60)
AddPerformance(Art Theatre, Don Juan, 28.12.2014 19:30, 1:45, 10)
AddPerformance(Theatre 199, Frozen, 18.01.2015 10:00, 1:00, 8)
AddPerformance(Theatre 199, Star Gate, 20.01.2015 22:00, 1:50, 6.45)
AddPerformance(Theatre 199, Don Juan, 18.01.2015 19:30, 2:00, 14.60)
AddPerformance(Theatre 199, I Pay, 18.01.2015 18:00, 1:30, 14.60)
PrintAllPerformances()
```

Sample Output

```
No performances
No theatres
Theatre added
Theatre added
Error: Duplicate theatre
Theatre 199, Theatre Sofia
Theatre added
Performance added
Error: Time/duration overlap
Error: Time/duration overlap
Performance added
Performance added
Error: Theatre does not exist
Performance added
Performance added
Performance added
Error: Time/duration overlap
(Frozen, Theatre 199,
```

```
PrintPerformances(Ivan Vazov National Theatre)
PrintPerformances(Art Theatre)
PrintPerformances(Theatre 199)


AddPerformance(Theatre 199, Long Story, 18.01.2015 11:00, 12:00, 100)
AddPerformance(Theatre 199, Short Story, 18.01.2015 19:40, 0:10, 0.30)
```

```
18.01.2015 10:00), (Don Juan,
Theatre 199, 18.01.2015
19:30), (Duende, Theatre 199,
20.01.2015 20:00), (Star
Gate, Theatre 199, 20.01.2015
22:00), (Bella Donna, Theatre
Sofia, 20.01.2015 19:30),
(Don Juan, Theatre Sofia,
20.01.2015 20:31)
No performances
Error: Theatre does not exist
(Frozen, 18.01.2015 10:00),
(Don Juan, 18.01.2015 19:30),
(Duende, 20.01.2015 20:00),
(Star Gate, 20.01.2015 22:00)
Error: Time/duration overlap
Error: Time/duration overlap
```

You are given the original source code from Huy Phuong designed to solve the above problem. Your task is to refactor it to improve its quality, fix any bugs and write some documentation.

Problem 1. Code Refactoring

Refactor the source code to improve its quality following the **best practices** introduced in the course "High-Quality Code". You are not allowed to modify the **`IPerformanceDatabase`** interface.

Some things to consider when refactoring:

- Renaming variables / methods / classes… to have meaningful and descriptive names
- Extracting each type into its own file
- Formatting the code well – the formatting should convey the code meaning and intent
- Splitting very long expressions into smaller pieces; introducing variables
- Reducing the complexity of control structures – for example, extracting too many nested control structures in their own methods
- Extracting methods; splitting long methods
- Extracting classes
- Proper validation and encapsulation; use of properties
- Proper usage of exceptions
- Proper usage of **`abstract`** and **`virtual`** members
- Ensuring all classes have strong cohesion – each class should do one thing only
- Reducing the coupling between classes
- Proper usage of OOP principles
- Following the principles of good OO design – SOLID, DRY, YAGNI, etc.
- Reducing the overall code complexity and making the code more readable
- Making the code testable

Any other refactorings are welcome if they improve the code quality. You may create as many classes, interfaces, enumerations, structures, etc. as you wish.

Note: If you are using ReSharper or a similar tool, beware that sometimes the refactorings it will try to make will actually make the code quality worse. Don't rely on it blindly; refactor the code so that you will like it and it will be easy to read, understand and maintain.

Problem 2. StyleCop

Make StyleCop run without any errors on your code (ignore all documentation-related errors).

Problem 3. Code Documentation

Document all the methods in the `IPerformanceDatabase` interface and the **`IPerformanceDatabase` interface declaration** using C# XML documentation. Any other documentation is **not** required.

For more information about the XML documentation tags and what should be included in it, you can look at the following MSDN article: https://msdn.microsoft.com/en-us/library/vstudio/b2s063f7%28v=vs.100%29.aspx.

You can see a list of recommended tags and examples on how to use them here: https://msdn.microsoft.com/en-us/library/vstudio/5ast78ax%28v=vs.100%29.aspx

Problem 4. Bug Fixing

Debug the code and fix any bugs you find. Be sure to test all special cases.

Be careful because you may introduce new bugs when refactoring the code. To ensure everything works as expected, test your code frequently. This should ideally be done via unit testing but testing the code "on the fly" will work too.

Problem 5. Unit Testing

Design and implement **unit tests for the following methods of the `IPerformanceDatabase` interface:**

- `ListTheatres()`
- `AddPerformance(string theatreName, string performanceTitle, DateTime startDateTime, TimeSpan duration, decimal price)`
- `ListPerformances(string theatreName)`

The **code coverage** should be **at least 90% for the specified methods**. The other methods are not required to be tested.

Follow the **Arrange / Act / Assert** pattern when writing unit tests.

Be sure to test **all major execution scenarios** + all interesting **border cases** and **special cases**. Use Visual Studio Team Test (VSTT) and VS code coverage.

Problem 6. * Mocking

Some methods cannot be tested on their own. **Use mocking** to make sure you pass the external dependencies when testing. Write unit tests for all methods in the **`IPerformanceDatabase`** interface.

In order to use mocking, you will have to make the code testable first. You will also have to find a way to pass external dependencies. One good way to do this is via the class constructor (the so-called **constructor-based dependency injection**). You can refer to the lectures for more examples (look at the **Mocking** lecture demos), or some other resource. A good starting point is this article in MSDN: https://msdn.microsoft.com/en-us/library/ff921152.aspx.

Problem 7. Correct Results in the Judge System

You are given an automated judge system to submit your solution (http://judge.softuni.bg/Contests/65/High-Quality-Code-23-Jan-2015). If your code is correct (all bugs are fixed) and runs fast enough (the performance bottlenecks are fixed), your solution will pass all the tests. The last 2 tests measure performance. The other tests measure correctness.

Use the **C# project/solution** option to upload your entire project. Submit your project as a .zip archive in the system. If everything is fine with your submission, you will soon receive feedback for your work.

If the file turns out to be too big, remove the **bin**, **obj** and **TestResults** folders.