# High-Quality Code Exam – Travel Agency

A junior developer Nashmat Abdal-Muhsin from Pakistan was once assigned to design and implement an information system for a travel agency. The original problem statement is given below:

А **travel agency** offers three types of tickets: air tickets, train tickets and bus tickets. **Air tickets** have flight number (e.g. NA2265), departure and arrival airports (e.g. London – Dubai), Airline Company (e.g. Northwest Airlines), departure date and time (e.g. 20.01.2015 17:25) and price (e.g. 1670.50). **Train tickets** have departure and arrival towns, departure date and time, regular price and decreased price for students. **Bus tickets** have departure and arrival towns, travelcompany, departure date and time and price. The travel agency should hold a **catalog of available tickets** and support searching by different criteria.

Model the travel agency and the tickets catalog with classes (following the best practices in the object-oriented programming) and write a program that executes arbitrary list of commands:

* **AddAir flight number; from; to; airline; date and time; price** – adds an air ticket to the catalog by given **flight number**, departure and arrival airports (**from**, **to**), **airline**, departure **date and time** and **price**. The flight number is unique and cannot be duplicated. As a result the command prints “**Ticket added**” or “**Duplicate ticket**” if such flight already exists.
* **DeleteAir flight number** – deletes an air ticket from the catalog by given **flight number**. As a result the command prints “**Ticket deleted**” or “**Ticket does not exist**” if the ticket could not be found in the catalog.
* **AddTrain from; to; date and time; regular price; students price** – adds a train ticket to the catalog by given departure and arrival towns (**from**, **to**), departure **date and time**, **regular price** and **students price**. The set of values (**from**, **to**, **date** and **time**) are unique and cannot be duplicated. As a result the command prints “**Ticket added**” or “**Duplicate ticket**” if such ticket already exists.
* **DeleteTrain from;** **to;** **date and** **time** – deletes a train ticket from the catalog by given departure town (**from**), arrival town (**to**) and **date and time**. As a result the command prints “**Ticket deleted**” or “**Ticket does not exist**” if the ticket could not be found in the catalog.
* **AddBus from; to; travel company; date and time; price** – adds a bus ticket to the catalog by given departure and arrival towns (**from**, **to**), **travel company**, departure **date and time** and **price**. The set of values (**from**, **to**, **travel company**, **date** and **time**) are unique and cannot be duplicated. As a result the command prints “**Ticket added**” or “**Duplicate ticket**” if such ticket already exists.
* **DeleteBus from;** **to;** **travel company;** **date and** **time** – deletes a bus ticket from the catalog by given departure town (**from**), arrival town (**to**), **travel company** and **date and time**. As a result the command prints “**Ticket deleted**” or “**Ticket does not exist**” if the ticket could not be found in the catalog.
* **FindTickets from; to** – finds all tickets from the catalog by given departure town/airport (**from**) and arrival town/airport (**to**). As a result the command prints all matching tickets on a single line, separated by spaces, in format **[date and time; type; price]** where **type** is either “**air**” or “**bus**” or “**train**” ordered by date and time (as first criteria, ascending), then by type (as second criteria, ascending) and then by price (as third criteria, ascending). In case of train tickets the regular price is printed and the student’s price is disregarded. Prices are always printed with exactly 2 digits after the decimal point. If no tickets are found by the specified criteria, the command prints “**Not found**”.
* **FindTicketsInInterval start date and time; end date and time** – finds all tickets from the catalog by given departure time interval (inclusive range). The result is the same like in the **FindTickets** command.

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

The **input** could contain up to 50 000 commands so the efficiently is important. All strings in the commands (e.g. flight number, name of town/airport, airline/travel companies) consist of alphabetical characters, numbers and spaces. Prices are real numbers with up to 2 digits after the decimal point, given as ‘.’ as decimal point separator (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. 22.01.2015 09:20). Empty lines in the input should be skipped.

|  |  |  |  |
| --- | --- | --- | --- |
| Sample Input  |  | | --- | | AddAir FX215; Sofia; Athens; Bulgaria Air; 17.01.2015 12:20; 200  AddAir SA522; Sofia; London; Lufthansa; 15.01.2015 07:15; 740  AddAir TX251FB; London; Paris; Swiss Air; 27.01.2015 05:00; 150  AddAir LU529; London; New York; Easyjet; 17.01.2015 11:00; 1200  AddAir TX251FB; London; Viena; Swiss Air; 29.01.2015 09:00; 150  AddBus Sofia; Varna; Biomet; 15.01.2015 06:15; 25  AddBus Sofia; Plovdiv; Vitosha; 27.01.2015 21:20; 12  AddBus Sofia; Plovdiv; Trakia Travel; 27.01.2015 21:20; 11.5  AddBus Sofia; Athens; Group Plus; 17.01.2015 02:00; 146.55  AddTrain Sofia; Rousse; 15.01.2015 09:20; 24.25; 12.15  AddTrain Varna; Sofia; 27.01.2015 05:22; 26.54; 12.22  AddTrain Sofia; Plovdiv; 17.01.2015 11:25; 6.5; 2.4  AddTrain Sofia; Athens; 15.01.2015 17:00; 124.5; 104  AddTrain Sofia; Plovdiv; 27.01.2015 21:20; 7.22; 2.61  AddTrain Sofia; Plovdiv; 25.01.2015 19:00; 7.22; 2.61  AddTrain Sofia; Plovdiv; 17.01.2015 11:25; 6; 2  AddBus Sofia; Plovdiv; Trakia Travel; 27.01.2015 21:20; 11  DeleteAir AA12245  DeleteBus Sofia; Athens; Group Plus; 17.01.2015 02:20;  DeleteTrain Sofia; Plovdiv; 25.01.2015 19:20  FindTickets Sofia; Plovdiv  FindTickets Sofia; Kaspichan  FindTicketsInInterval 15.01.2015 06:00; 15.01.2015 09:20  DeleteAir SA522  FindTicketsInInterval 15.01.2015 06:00; 15.01.2015 09:20  FindTicketsInInterval 17.01.2015 12:30; 17.01.2015 19:00 | | Sample Output  |  | | --- | | Ticket added  Ticket added  Ticket added  Ticket added  Duplicate ticket  Ticket added  Ticket added  Ticket added  Ticket added  Ticket added  Ticket added  Ticket added  Ticket added  Ticket added  Ticket added  Duplicate ticket  Duplicate ticket  Ticket does not exist  Ticket does not exist  Ticket does not exist  [17.01.2015 11:25; train; 6.50] [25.01.2015 19:00; train; 7.22] [27.01.2015 21:20; bus; 11.50] [27.01.2015 21:20; bus; 12.00] [27.01.2015 21:20; train; 7.22]  Not found  [15.01.2015 06:15; bus; 25.00] [15.01.2015 07:15; air; 740.00] [15.01.2015 09:20; train; 24.25]  Ticket deleted  [15.01.2015 06:15; bus; 25.00] [15.01.2015 09:20; train; 24.25]  Not found | |

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

## Problem 1. Code Refactoring

**Refactor the source code** to improve its quality following the best practices introduced in the course  
“[High-Quality Code](https://softuni.bg/courses/high-quality-code/)”. You are not allowed to modify the ITicketCatalog interface. Any other refactorings are welcome if they improve the code quality.

**36 score**

## Problem 2. Bug Fixing

**Debug the code**, **find and fix the bugs** in it.

**6 score**

## Problem 3. Unit Testing

Design and implement **unit tests for** **all methods of the** ITicketCatalog **interface**. Any other code is not required to be tested. The **code coverage** should be **at** **least 90% for the classes implementing** ITicketCatalog (you do not need to cover the class that parses the input commands and prints the output). 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.

**30 score**

## Problem 4. Code Documentation

**Document the methods** AddAirTicket**,** DeleteBusTicket**,** FindTickets **and** FindTicketsInIntervaldefined in **the** ITicketCatalog **interface** and the ITicketCatalog **interface declaration** using C# XML documentation. Any other documentation is **not** required. Five documentations 🡪 each gives 1 score.

**5 score**

## Problem 5. Performance Bottlenecks

Find any **performance bottlenecks** and briefly describe them with a **comment in the code**.

**Fix the problems** if possible (and leave the bottlenecks descriptions in addition to the fixes).

**7 score**

## Problem 6. Correct Results in the Judge System

You are given an automated judge system to submit your solution (<http://judge.softuni.bg/Contests/61/High-Quality-Code-19-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 others measure correctness.

**16 score**