# High-Quality Code Exam – Ticket Office

A junior developer Salimur Mamun Rahman from Bangladesh was once assigned to design and implement an information system for a **ticket office**. The original problem statement is given below:

А **ticket office** offers three types of tickets: bus tickets, train tickets and air tickets. **Bus tickets** have departure and arrival towns, travel company, departure date and time and price. **Train tickets** have departure and arrival towns, departure date and time, regular price and decreased price for students. **Air tickets** have flight number (e.g. FB1364), departure and arrival airports (e.g. London – Dubai), Airline Company (e.g. Lufthansa), departure date and time (e.g. 18.01.2015 17:25) and price (e.g. 1870.50). The ticket office should hold a **repository of available tickets** and support searching by different criteria.

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

* **CreateBus from|to|travel\_company|date\_and\_time|price** – adds a bus ticket to the repository 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 “**Bus created**” or “**Duplicated bus**” if such bus already exists.
* **DeleteBus from|to|travel\_company|date\_and**\_**time** – deletes a bus ticket from the repository by given departure town (**from**), arrival town (**to**), **travel company** and **date and time**. As a result the command prints “**Bus deleted**” or “**Bus does not exist**” if the bus could not be found in the repository.
* **CreateTrain from|to|date\_and\_time|regular\_price|student\_price** – adds a train ticket to the repository by given departure and arrival towns (**from**, **to**), departure **date and time**, **regular price** and **student price**. The set of values (**from**, **to**, **date** and **time**) are unique and cannot be duplicated. As a result the command prints “**Train created**” or “**Duplicated train**” if such train already exists.
* **DeleteTrain from|to|date\_and**\_**time** – deletes a train ticket from the repository by given departure town (**from**), arrival town (**to**) and **date and time**. As a result the command prints “**Train deleted**” or “**Train does not exist**” if the train could not be found in the repository.
* **CreateFlight flight\_number|from|to|airline|date\_and\_time|price** – adds a flight ticket to the repository 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 “**Flight created**” or “**Duplicated flight**” if such flight already exists.
* **DeleteFlight flight\_number** – deletes a flight ticket from the repository by given **flight number**. As a result the command prints “**Flight deleted**” or “**Flight does not exist**” if the flight could not be found in the repository.
* **FindTickets from|to** – finds in the repository all tickets from given departure town/airport (**from**) to given 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 “**flight**” 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 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 “**No matches**”.
* **FindByDates start\_date\_and\_time|end\_date\_and\_time** – finds all tickets from the repository by given departure time interval (inclusive range). The result is the same like in the **Find Tickets** 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  |  | | --- | | CreateBus Sofia|Varna|Etap Address|15.01.2015 06:15|25  CreateBus Sofia|Varna|Vitosha Travel|27.01.2015 21:20|12  CreateBus Sofia|Varna|Black Sea Travel|27.01.2015 21:20|11.5  CreateBus Sofia|Athens|Arda Tour|17.01.2015 02:00|146.55  CreateBus Sofia|Varna|Black Sea Travel|27.01.2015 21:20|11  CreateTrain Sofia|Rousse|15.01.2015 09:20|24.25|12.15  CreateTrain Varna|Sofia|27.01.2015 05:22|26.54|12.22  CreateTrain Sofia|Varna|17.01.2015 11:25|6.5|2.4  CreateTrain Sofia|Athens|15.01.2015 17:00|124.5|104  CreateTrain Sofia|Varna|27.01.2015 21:20|7.22|2.61  CreateTrain Sofia|Varna|25.01.2015 19:00|7.22|2.61  CreateTrain Sofia|Varna|17.01.2015 11:25|6|2  CreateFlight FX215|Sofia|Varna|Bulgaria Air|17.01.2015 12:20|200  CreateFlight SA522|Sofia|London|Lufthansa|15.01.2015 07:15|740  CreateFlight TX251FB|London|Paris|Swiss Air|27.01.2015 05:00|150  CreateFlight LU529|London|New York|Easyjet|17.01.2015 11:00|1200  CreateFlight TX251FB|London|Viena|Swiss Air|29.01.2015 09:00|150  DeleteFlight AA12245  DeleteBus Sofia|Athens|Arda Tour|17.01.2015 02:20|  DeleteTrain Sofia|Varna|25.01.2015 19:20  FindTickets Sofia|Varna  FindTickets Sofia|Plovdiv  FindByDates 15.01.2015 06:00|15.01.2015 09:20  DeleteFlight SA522  FindByDates 15.01.2015 06:00|15.01.2015 09:20  FindByDates 17.01.2015 12:30|17.01.2015 19:00 | | Sample Output  |  | | --- | | Bus created  Bus created  Bus created  Bus created  Duplicated bus  Train created  Train created  Train created  Train created  Train created  Train created  Duplicated train  Flight created  Flight created  Flight created  Flight created  Duplicated flight  Flight does not exist  Bus does not exist  Train does not exist  [15.01.2015 06:15|bus|25.00] [17.01.2015 11:25|train|6.50] [17.01.2015 12:20|flight|200.00] [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]  No matches  [15.01.2015 06:15|bus|25.00] [15.01.2015 07:15|flight|740.00] [15.01.2015 09:20|train|24.25]  Flight deleted  [15.01.2015 06:15|bus|25.00] [15.01.2015 09:20|train|24.25]  No matches | |

You are given the original source code from Salimur 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 ITicketRepository 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** ITicketRepository **interface**. Any other code is not required to be tested. The **code coverage** should be **at** **least 90% for the classes implementing** ITicketRepository (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** ITicketRepository **interface** and the ITicketRepository **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/62/>). 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**