# Упражнения: Комбиниране на структури от данни

Тествайте решението в Judge: <https://judge.softuni.org/Contests/3602/Combining-Data-Structures-Exercises>

Използвайте дадения скелет:

Graphical user interface, text, application

Description automatically generated

## Колекция от People

* bool AddPerson**(string email, string name, int age, string town)**
  + Имейлите са **уникални**.
  + Ако има имейл, който съществува, върнете **false** (без да добавите нов обект Person), в противен случай върнете **true**
* Person FindPerson**(string email)**
  + Връща обект **Person** или **null** (ако няма такъв)
* Bool DeletePerson**(string email)**
  + Връща **true** (ако е успешно изтрит) или **false** (ако не е намерен).
* IEnumerable<Person> FindPeople**(string emailDomain)**
  + Връща **колекция от хора** с еднакъв **домейн**, сортирани по **имейл**
* IEnumerable<Person> FindPeople**(string name, string town)**

Връща **колекция от хора** с еднакво **име и град**, сортирани по имейл

Нека първо да разгледаме класа Person. Има няколко свойства и конструктор:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Следващият клас PersonCollection има недовършени имплементации на методите за структурата:

Graphical user interface, application

Description automatically generated

Проектът идва с **компонентни тестове** и **тестове за производителност**, който проверяват функционалността **на структурата от данни** “**person collection**”.

Text

Description automatically generated

Нека да имплементираме решението, което използва **ефективни базови структури от данни**.

Първо трябва да **дефинираме** **структурите от данни**, за да изпълним **ефективно** операциите:

* За да **намерим person чрез имейл** може да използваме **речник**. Очакваме да има най-много един човек с такъв имейл (припомнете си, че имейлът е **уникален**)



* За да намерим всички обекти **person** с еднакви имейл **домейни**, може да използваме **речник**. Ще използваме имейл домейна като **ключ**, а като стойност **сортиран сет от person**:



* За да намерим всички обекти person по **име и град** може да използваме **хеш таблица**. Може да комбинираме **името и града като един низ** и да го използваме като **ключ** и **сортиран сет** от person като **стойност**:



Имаме **три** отделни структури от данни, които работят заедно, за да се постигне **ефективност** на операциите на структурата от данни “person collection”. Когато **комбинираме** структури от данни, винаги трябва да поддържаме структурите от данни **актуални**:

* Add() трябва да **добавя стойност** за всички структури от данни.
* Modify() трябва да **актуализира** всички структури от данни, за да съдържа правилни данни.
* Delete() трябва да **изтрие** данните от всичките структури от данни.

Вече сме готови да имплементираме операциите, базирани на структурите от данни по-горе.

Не забравяйте да **инициализирате** структурите от данни:

Graphical user interface, text, timeline

Description automatically generated

Първо трябва да имплементираме **add**, **edit** и **delete**.

Преди да направи това, трябва да поговорим за **допълнителните методи**, които ще използваме. Ще трябва да опростим работата с **речници**, съдържащи **набори от стойности**. Тези методи са **generic**, защото искаме да ги използваме за всякакъв вид речници, съдържащи колекции от стойности.

Разширените методи в C# осигуряват специален синтаксис за добавяне на методи към клас **без да бъде наследяван или променян**. Методите **Generic в C#** позволяват типовете данни на **входните** и **изходните** параметри на метода да бъдат **общи** - типовете данни стават **параметри**. Комбинирането на **разширените** и **generic** методи може да разшири речниците в C# (IDictionary interface) и да добави полезни методи за спестяване на код.

Погледнете класа DictionaryExtensions в основния проект. Осигурява няколко разширени метода за речниците

Метода AppendValueToKey(key, value) **добавя стойност към колекция от ключове**, свързани към **централен ключ**. Ако този ключ **не съществува**, първо се **създава колекцията**:

Graphical user interface, text, application

Description automatically generated

Друг полезен разширен метод за речници, съдържащи колекция от стойности, е GetValuesForKey(key). Той връща **всички стойности** за определен ключ за речника. Приема се, че ключът съдържа колекция от стойности или нищо, ако не съществува.

Graphical user interface, text, application

Description automatically generated

### AddPerson()

Нека да напишем метода AddPerson(). Трябва да върнем **false**, ако вече имаме такъв **имейл адрес**:

Graphical user interface, text, application, email

Description automatically generated

Ако няма, ще **създадем** и ще **добавим** нов person:

Graphical user interface, text, application, email

Description automatically generated

Можем да вземем имейл домейна с разделяне и да запълним и другите структури:Text

Description automatically generated

### Count

След това да имплементираме свойството Count, за да взимаме броя на хората във всяка колекция (броят трябва да бъде еднакъв):

Text

Description automatically generated

### FindPerson()

Следващият метод е FindPerson(email). В него трябва да използваме метода TryGetValue(), защото може да **няма** такъв имейл в нашия речник:

Graphical user interface, text, application

Description automatically generated

### DeletePerson()

Методът DeletePerson(email) е подобен на AddPerson(). Опитайте се да намерите човек. Ако **не съществува**, върнете **false**:

Graphical user interface, text, application

Description automatically generated

След това **премахваме по имейл**. Намираме **домейна** и го премахваме. Накрая премахваме по град.

Text

Description automatically generated

Стартираме **компонентните тестовете** – някои ще бъдат успешни, други няма да бъдат:

Text

Description automatically generated

### FindPeople() by Domain

Нека сега да имплементираме метода FindPeople**(emailDomain)**.

Гарантирано е, че ключът emailDomain е в речника personsByEmailDomain, затова не трябва да правим допълнителна проверка. Ключът гарантирано ще бъде **създаден**, когато person се създава в метода **AddPerson()**.

Методът е подобен на **метода за намиране**. Този път може да използваме нашия **разширен метод в речника**:

Graphical user interface, text, application

Description automatically generated

Стартираме **компонентните тестовете** отново:

Text

Description automatically generated

Още един успешен тест!

### FindPeople() по име и град

Сега трябва да имплементираме **метода** the FindPeople**(name, town)**. Може да използваме разширения метод GetValuesForKey(), за да комбинираме **име и град**. Този комбиниран ключ се търси в базовия речник personsByNameAndTown, който връща колекция от хора.

Graphical user interface, text, application

Description automatically generated

Сега трябва да минат всички тестове:

Text

Description automatically generated

**Поздравления**! Вие успешно имплементирахте структурата от данни “person collection”.

## Търговския център

Търговският център съдържа набор от продукти. Всеки продукт има **име**, **цена** и **производител**. Вашата задача е да моделирате търговския център и да направите **структура от данни**, която да съдържа продуктите. Напишете програма, която чете **N** команди.

* string AddProduct(string name, decimal price, string producer)
  + **Добавя** продукт с дадено **име**, **цена** и **производител**. Позволени са **дублиранията**. Трябва да отпечатате **„Product added“** като резултат.
* string DeleteProducts(string producer)
  + **Изтрива** всички продукти със **съвпадащ** **производител**. Отпечатайте “**X products deleted**”, където **X** е броят на изтритите продукти.
  + Отпечатайте “**No products found**”, ако **няма** такива продукти.
* string DeleteProducts(string name, string producer)
  + Изтрива всички продукти със съвпадащо **име** и **производител**. Отпечатайте “**X products deleted**”, където **X** е броят на **изтритите** продукти.
  + Отпечатайте “**No products found**”, ако **няма** такива продукти.
* string FindProductsByName(string name)
  + **Намираме** всички продукти по дадено **име**. Отпечатайте списък от продукти във формата: **{име;производител;цена}**, сортирайте ги по **име**, **производител** и **цена**.
  + Ако **няма** продукти със съвпадащо име, отпечатайте “**No products found**”
* string FindProductsByProducer(string producer)
  + Намерете всички продукти от даден **производител**. Отпечатайте списък от продукти във формата **{име;производител;цена}**, сортирайте ги по **име**, **производител** и **цена**.
  + Ако **няма** продукти със съвпадащ производител, отпечатайте “**No products found**”
* string FindProductsByPriceRange(decimal fromPrice, decimal toPrice)
  + **Намерете** всички продукти, които имат цена, **по-голяма или равна** на **fromPrice** и **по-малка или равна** на **toPrice**.
  + Отпечатайте списък с продуктите във формата: **{име;производител;цена}**, сортирайте ги по **име**, **производител** и **цена**.
  + Ако **няма** продукти с такъв обхват на цената, отпечатайте “**No products found**”

Всички операции за съвпадение на низове са чувствителни към **малки и главни букви** (case sensitive).

### Вход

Входът трябва да се **чете от конзолата**.

* На **първия** ред ще получите броя на командите, **числото N**.
* На следващите **N реда** ще получите **команди** във формата по-горе:

Входът винаги ще бъде **валиден** и във **формата по-горе**. Няма нужда да проверявате.

### Изход

Изходът трябва да се **отпечата на конзолата**.

Изходът трябва да съдържа изхода от всяка команда във входа

### Бележки

* **N** ще бъде между **1** и **50 000**, включително.
* Всички низове в командите (примерно производител и име на продукт) съдържат **букви от азбуката, числа и интервали**. Низовете са чувствителни към **малки и главни букви**.
* Цените, които се четат на входа, са реални числа с **до 2 цифри** след десетичната запетая (примерно 133.58, 320.3, или 10)
* Символът ‘.’ се използва за **десетична запетая**.
* Цените в изхода трябва да са с **точно 2 цифри** след десетичната запетая (Примерно 320.20 вместо 320.2)

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 17  AddProduct IdeaPad Z560;1536.50;Lenovo  AddProduct ThinkPad T410;3000;Lenovo  AddProduct VAIO Z13;4099.99;Sony  AddProduct CLS 63 AMG;200000;Mercedes  FindProductsByName CLS 63 AMG  FindProductsByName CLS 63  FindProductsByName cls 63 amg  AddProduct 320i;10000;BMW  FindProductsByName 320i  AddProduct G560;999;Lenovo  FindProductsByProducer Lenovo  DeleteProducts Lenovo  FindProductsByProducer Lenovo  FindProductsByPriceRange 100000;200000  DeleteProducts Beer;Ariana  DeleteProducts CLS 63 AMG;Mercedes  FindProductsByName CLS 63 AMG | Product added  Product added  Product added  Product added  {CLS 63 AMG;Mercedes;200000.00}  No products found  No products found  Product added  {320i;BMW;10000.00}  Product added  {G560;Lenovo;999.00}  {IdeaPad Z560;Lenovo;1536.50}  {ThinkPad T410;Lenovo;3000.00}  3 products deleted  No products found  {CLS 63 AMG;Mercedes;200000.00}  No products found  1 products deleted  No products found |

### Насоки

В проекта има **4 класа**.

Text

Description automatically generated

Класът ShoppingStructure е **празен**. Там трябва да **имплементирате методите** за структурата от данни.

Класът DictionaryExtensions има **разширени методи**, които може да използвате, ако искате да използвате **сложни речници**:

Graphical user interface, text, application

Description automatically generated

Класа Product е **имплементиран**. Може да се използва при **добавяне** във вашата структура:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Последният клас, **StartUp**, е мястото, където се изпълнява кодът и където ще трябва да **обработвате всяка команда**:

Text

Description automatically generatedGraphical user interface, text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text

Description automatically generated