

# Упражнения: Компаратори

## Problem 1. Сравнима книга

Разширете решението на предишната задача. Имплементирайте интерфейса **Comparable<Book>** в съществуващия клас **Book**. Сравнението на две книги трябва да се случва по следния ред:

- Първо ги сортирайте по **възходящ хронологичен** ред (по година)
- Ако две книги са публикувани в **една и съща година**, сортирайте ги **по азбучен ред**

Предефинирайте метода **ToString()** в своя клас **Book**, за да връща низ във формата:

- {заглавие} - {година}

Променете своя клас **Library** така, че да съхранява книгите в правилния ред.

Не е нужно да променяте нищо в своя **Main** метод от предишната задача освен начина на отпечатване на книга в конзолата.

## Примери

Изход
The Documents in the Case - 1930
The Documents in the Case - 2002
Animal Farm - 2003

## Решение

```
public class Book : Comparable<Book>
{
    public Book(string title, int year, params string[] authors)
    {
        Title = title;
        Year = year;
        Authors = authors;
    }

    public string Title { get; private set; }
    public int Year { get; private set; }
    public IReadOnlyList<string> Authors { get; private set; }

    public int CompareTo(Book other)
    {
        int result = this.Year.CompareTo(other.Year);
        if (result == 0)
        {
            result = this.Title.CompareTo(other.Title);
        }
        return result;
    }

    public override string ToString()
    {
        return $"{this.Title} - {this.Year}";
    }
}
```

## Problem 2. Сравняващият книги

Разширете решението на предишната задача. Създайте клас **BookComparator**, който да имплементира интерфейса **IComparer<Book>** и така да включва следния метод:

- **int Compare(Book, Book)**

**BookComparator** трябва да сравнява две книги по:

1. Заглавие – азбучен ред
2. Година на издаване на книгата – от най-нови към най-стари

Модифицирайте своя клас **Library** отново, така че да имплементирате **новото сортиране**.

### Примери

Startup.cs
<pre>public static void Main() {     Book bookOne = new Book("Animal Farm", 2003, "George Orwell");     Book bookTwo = new Book("The Documents in the Case", 2002, "Dorothy Sayers", "Robert Eustace");     Book bookThree = new Book("The Documents in the Case", 1930); }</pre>

Изход
Animal Farm - 2003 The Documents in the Case - 2002 The Documents in the Case - 1930

### Решение

```
public class BookComparator : IComparer<Book>
{
    public int Compare(Book x, Book y)
    {
        int result = x.Title.CompareTo(y.Title);
        if (result == 0)
        {
            result = y.Year.CompareTo(x.Year);
        }

        return result;
    }
}
```

## Problem 3. Сравняване на обекти

Има нещо такова като интерфейс **Comparable**, предполагам, че вероятно вече го знаете. Вашата задача е проста. Създайте клас **Person**. Всеки човек трябва да има **име**, **възраст** и **град**. Трябва да имплементирате интерфейса **IComparable<T>** и метода **CompareTo**. Когато сравнявате двама души, първо **сравнете имената им**, след това – **възрастите им**, а накрая – **градовете им**.

## Вход

На всеки ред ще получавате човек във формат:

{име} {възраст} {град}

Колекционирайте ги, докато не получите "END"

След това ще получите цяло число **N** - **N<sup>тия</sup>** човек в колекцията ви. **Започвайки от 1.**

## Изход

На единствения ред от изхода изведете статистики: колко хора са еднакви с него, колко не са и общият брой хора във вашата колекция.

Формат: {брой еднакви хора} {брой нееднакви хора} {общ брой хора}

## Ограничения

Входните имена, възрасти и адреси ще са валидни. Входното число винаги ще е валидно цяло число в интервала [2...100]

Ако няма еднакви хора, отпечатайте: "No matches"

## Примери

Вход	Изход
Pesho 22 Vraca Gogo 14 Sofeto END 2	No matches
Pesho 22 Vraca Gogo 22 Vraca Gogo 22 Vraca END 2	2 1 3

## Problem 4. Шаблон Strategy

Интересен шаблон, за който може да сте чували, е **Strategy**; ако има няколко начина да се изпълни задача (например **да се сортира колекция**), той позволява на клиента да **избере начина, който най-много подхожда на нуждите му**. Известна имплементация на шаблона в C# са методите [List<T>.Sort\(\)](#) и [Array.Sort\(\)](#), които използват **IComparer** като аргумент.

Създайте клас **Person**, който съдържа **име** и **възраст**. Създайте два компаратора за Person (класове, които имплементират интерфейса **IComparer<Person>**). Първият компаратор трябва да сравнява хора по **дължината на името им** като първи параметър; ако двама души имат имена с една и съща дължина, вместо това сравнява **първата буква** от имената, **без да прави разлика между малки и главни букви**. Вторият компаратор трябва да ги сравнява по **възраст**.

Създайте 2 обекта от тип **SortedSets** с елементи от типа **Person**; първият трябва да имплементира **компараторът за имена**, а втория да имплементира **компаратора за възраст**.

## Вход

На първия ред от входа ще получите число **N**. На всеки от следващите **N** реда ще получите информация за хора във формата “<name> <age>”. Добавете хората от входа и в двете сортирани колекции (те трябва да съдържат всички хора, подадени като входни данни).

## Изход

Обходете с foreach колекциите и отпечатайте всеки човек от тях на нов ред в същия формат, в който сте го получили. Започнете с този, който имплементира компаратора за имена.

## Ограничения

- Името на човек ще е низ, който съдържа само букви и цифри и ще е с дължина [1...50] символа.
- Възрастта на човек ще е положително цяло число между [1...100].
- Броят хора **N** ще е положително цяло число между [0...100].

## Примери

Вход	Изход
3 Pesho 20 Joro 100 Pencho 1	Joro 100 Pesho 20 Pencho 1 Pencho 1 Pesho 20 Joro 100
5 Ivan 17 asen 33 Stoqn 25 Nasko 99 Joro 3	asen 33 Ivan 17 Joro 3 Nasko 99 Stoqn 25 Joro 3 Ivan 17 Stoqn 25 asen 33 Nasko 99

## Problem 5. \*Логика за еднаквост

Създайте клас **Person**, съдържащ **име** и **възраст**. Хора с еднакви име и възраст възприемайте за един и същи; предефинирайте всички необходими методи, за да наложите тази логика. Вашият клас трябва да работи както със стандартни, така и с хеширани колекции. Създайте **SortedSet** и **HashSet** от типа Person.

## Вход

На първия ред от входа ще получите число **N**. На всеки от следващите **N** реда ще получите информация за хора във формата “<name> <age>”. Добавете хората от входа и в двете колекции (те трябва да съдържат всички хора, получени като информация от входа).

## Изход

Изходът трябва да се състои от точно два реда. На първия трябва да отпечатате размера на дървовидната колекция, а на втория – размера на хешираната.

## Ограничения

- Името на човек ще е низ, който съдържа само букви и цифри и ще е с дължина [1...50] символа.
- Възрастта на човек ще е положително цяло число между [1...100].
- Броят хора **N** ще е положително цяло число между [0...100].

## Примери

Вход	Изход
4 Pesho 20 Peshp 20 Joro 15 Pesho 21	4 4
7 Ivan 17 ivan 17 Stoqn 25 Ivan 18 Ivan 17 Stopn 25 Stoqn 25	5 5

## Съвет

Трябва да предефинирате и метода **Equals**, и метода **GetHashCode**. Можете да потърсите имплементация на **GetHashCode** онлайн – не е нужно да е съвършена, но трябва да е достатъчно добра да произведе същия хеш код за обекти с еднакви име и възраст, както и достатъчно различни хеш кодове за обекти с различни име и/или възраст.

## Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма **"Обучение за ИТ кариера"** на МОН за подготовка по професия "Приложен програмист".



- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под **свободен лиценз CC-BY-NC-SA** (Creative Commons Attribution-Non-Commercial-Share-Alike 4.0 International).

