МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота № 5

з дисципліни “Основи програмування”
тема “Об'єкти і класи”

Виконав
студент І курсу
групи КП-01
Пецеля Артем Володимирович
(*прізвище, ім’я, по батькові*)

Перевірив
“____” “_____” 20___ р.
викладач
Гадиняк Руслан Анатолійович
(*прізвище, ім’я, по батькові*)

варіант №12

Київ 2021

Мета роботи

Створення і використання класів для об'єктів з даними без логіки та об'єктів динамічних колекцій елементів.

Генерування та порядкова обробка великої кількості даних у форматі CSV.

Постановка завдання

Частина 1. Генератор CSV

Створити консольну програму, що на основі двох заданих аргументів командного рядка генерує у текстовий файл дані у форматі CSV. Перший аргумент задає шлях до файлу, другий - кількість рядків у CSV таблиці з даними, які будуть згенеровані. Наприклад, `dotnet run ./out.csv 13`. Якщо не задано одного з аргументів, або у аргументах є помилка - друкувати про це повідомлення в консолі і завершувати роботу програми. До аргументу з назвою файлу не додавати в програмі жодних додаткових рядків, не робити обмеження лише на `.csv` розширення, воно може бути будь-яким і бути відсутнім. Кількість рядків даних - невід’ємне число. Дані, які генеруються у файл описують сутності із лабораторної роботи №5 минулого семестру. CSV файл має містити перший рядок із назвами стовпців (не входить в кількість рядків, які потрібно генерувати). Ідентифікатори генерувати унікальними в рамках файлу. Для генерації рядкових даних можна задати масив з рядками і випадковим чином вибирати звідти рядки, їх можна склеювати.

Достатньо зберігати дані без CSV екранування, якщо вони будуть генеруватись з відповідними обмеженнями на дані.

Частина 2.

Створити консольну програму, що зчитує CSV дані заданих за варіантом сутностей з 2-х файлів, виконує їх перетворення і записує результат як CSV у 3-ій файл.

Попередньо згенерувати вхідні файли за допомогою програми-генератора з першої частини завдання. Перший з файлів має містити близько 10-20 сутностей, другий - 100000-200000 сутностей.
Шляхи до 3-х файлів достатньо задати в коді (але 1 раз в головній функції) без аргументів командного рядка.

Перенести у код роботи структуру даних із лабораторної роботи №5 і зробити її класом. Додати у клас два конструктора: без параметрів і з параметрами для всіх полів. Додати в клас реалізацію стандартного метода ToString (див. додаток).

Створити клас (ListEntity, Entity замінити на назву типу сутност) для списку сутностей за варіантом на основі масиву об’єктів (див. додаток).

Реалізувати функцію (замінити entity і entities на назву сутності в однині і множині, за варіантом):
static ListEntity ReadAllEntities(string filePath);

У цій функції зчитувати текст з CSV файлу порядково (див додатки), у об’єкт власної реалізації списка із об’єктами типу сутності за варіантом. Файл можна зчитати лише один раз.
Перевіряти рядки CSV, якщо у них помилка (наприклад, кількість даних у рядку не рівна кількості стовпців) викидати помилку, що завершить роботу програми.

Використати ReadAllEntities для обох вхідних файлів і вивести кількість зчитаних рядків даних і перші 10 рядків даних (якщо є) у консоль.
Створити новий об’єкт ListEntity і переписати в нього всі елементи з обох списків так, щоби в новому списку не повторювались ідентифікатори

сутностей (всі ідентифікатори були унікальні). Тобто, не переписувати у новий список сутність, якщо у списку вже є інша сутність з таким ідентифікатором.
Обрати числове поле вашого типу сутності (але не ідентифікатор) і знайти по списку середнє арифметичне значення по цьому полю.
Видалити зі списку всі об’єкти, значення відповідного поля яких менше за знайдене середнє арифметичне.
Створити для цих дій окремі функції.

Реалізувати функцію (замінити entity і entities на назву сутності в однині і множині, за варіантом):
static void WriteAllEntities(string filePath, ListEntity entities);

Записати результат (модифікований третій список) у вихідний файл в форматі CSV порядково (див додатки).

Обмеження

- При реалізації завдань заборонено:

використання стандартних типів колекцій і інших алгоритмів, що дані у завданні, їх поведінку необхідно реалізувати самостійно.
використання статичних полів.

- Функції мають мати одне призначення, без зайвих параметрів, повертати результат там, де це можливо.

Назви функцій, типів, полів і змінних мають слідувати одному стилю і відповідати їх призначенню.

Аналіз вимог і проектування

CSV-текст з даними столиць:

```
0;Baku;927011;213,366
1;Vienna;927562;32,949
3;Ouagadougou;612395;118,108
5;Sofia;948723;545,533
6;Sucre;862313;192,736
8;Sucre;839487;77,031
10;Ottawa;639897;654,697
```

Таблиця рядків даних:

0	Baku	927011	213,366
1	Vienna	927562	32,949
3	Ouagadougou	612395	118,108
5	Sofia	948723	545,533
6	Sucre	862313	192,736
8	Sucre	839487	77,031
10	Ottawa	639897	654,697

Структура даних “Столиці” була розроблена з такими полями:

1. id - ідентифікатор поля
2. name - назва столиці
3. country - країна, в якій це місто є столицею
4. population - кількість населення в мільйонах
5. area - площа міста в кілометрах квадратних.

Тексти коду програм

./lab1Part1/Program.cs

```
using System;
using static System.Console;
using static System.IO.File;

namespace lab1
{
    class StringBuilder
    {
        private string[] strings;
        private int size;

        public StringBuilder()
        {
            strings = new string[16];
            size = 0;
        }
        public StringBuilder Append(string str)
        {
            if (strings.Length == size)
            {
                Expand();
            }
            strings[size] = str;
            size += 1;
            return this;
        }
        private void Expand()
        {
            int oldCapacity = strings.Length;
            string[] oldArray = strings;
            strings = new string[oldCapacity * 2];
            Array.Copy(oldArray, strings, oldCapacity);
        }
        private int GetTotalLength()
        {
            int charCounter = 0;
            for (int i = 0; i < size; i++)
            {
                string str = strings[i];
                charCounter += str.Length;
            }
            return charCounter;
        }
        public override string ToString()
        {
            int charCounter = GetTotalLength();
            char[] buffer = new char[charCounter];
            int index = 0;
            for (int i = 0; i < size; i++)
            {
                string str = strings[i];
                Array.Copy(str.ToCharArray(), 0, buffer, index, str.Length);
                index += str.Length;
            }
            string allStrings = new string(buffer);
            return allStrings;
        }
    }

    class Program
    {
        struct Options
        {
            public string parsingError;
            public string outputFile;
            public int numberOfCapitals;
```

```
        }
        static Options ParseOptions(string[] args)
        {
            Options options = new Options();
            if (args.Length < 2)
            {
                options.parsingError = "Not enough arguments";
            }
            else if (args.Length > 2)
            {
                options.parsingError = "Too much arguments";
            }
            else if (!int.TryParse(args[1], out options.numberOfCapitals))
            {
                options.parsingError = "The third argument is not a number";
            }
            else if (options.numberOfCapitals < 0)
            {
                options.parsingError = "Number is negative";
            }
            else
            {
                options.outputFile = args[0];
            }
            return options;
        }
        struct Capital
        {
            public int id;
            public string name;
            public string country;
            public int population;
            public double area;
        }
        static Capital[] GenerateCapitals(int N)
        {
            Capital[] capitals = new Capital[N];
            Random rand = new Random();
            string[] namesPattern = new string[] { "Canberra", "Vienna", "Baku", "Manama", "Sucre", "La P
az", "Brasília", "Sofia", "Ouagadougou", "Ottawa", "West Island",
            "Moroni", "Prague", "Copenhagen", "Hanga Roa", "Asmara", "Helsinki", "Papeete", "Tbilisi", "C
onakry", "New Delhi", "Jerusalem"};
            string[] countryPattern = new string[] {"Bahrain", "Bangladesh", "Barbados", "Belarus", "Belg
ium", "Belize", "Benin", "Bhutan", "Bolivia", "Bosnia Herzegovina",
            "Botswana", "Brazil", "Brunei", "Bulgaria", "Burkina", "Burundi", "Cambodia", "Cameroon", "Ca
nada", "Cape Verde", "Central African Rep" };

            for (int i = 0; i < capitals.Length; i++)
            {
                capitals[i].id = i;
                capitals[i].name = namesPattern[rand.Next(namesPattern.Length)];
                capitals[i].country = countryPattern[rand.Next(countryPattern.Length)];
                capitals[i].population = rand.Next(1000000);
                capitals[i].area = Math.Round(rand.Next(1000) * rand.NextDouble(), 3);
            }
            return capitals;
        }
        static string MakeCsvString(Capital[] capitals)
        {
            StringBuilder sb = new StringBuilder();
            sb.Append("id;name;country;population;area\n");
            for (int i = 0; i < capitals.Length; i++)
            {
                sb.Append(capitals[i].id.ToString()).Append(";").Append(capitals[i].name).Append(";").App
end(capitals[i].country).Append(";").Append(capitals[i].population.ToString()).Append(";").Append(capital
s[i].area.ToString());
                if (i != capitals.Length - 1)
                {
                    sb.Append("\n");
                }
            }
            return sb.ToString();
        }
```

```
        static void Main(string[] args)
        {
            Options options = ParseOptions(args);
            if (options.parsingError != null)
            {
                WriteLine("Parsing error: {0}", options.parsingError);
                return;
            }
            Capital[] capitals = GenerateCapitals(options.numberOfCapitals);
            string csvString = MakeCsvString(capitals);
            WriteAllText(options.outputFile, csvString);
            WriteLine("Program has genereted {0} capitals succesfully", options.numberOfCapitals);
        }
    }
}
```

./lab1Part2/Program.cs

```
using System;
using System.Diagnostics;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using static System.Console;

namespace lab1Part2
{
    class Capital
    {
        private int id;
        private string name;
        private string country;
        private int population;
        private double area;
        public Capital()
        {

        }
        public Capital(int id, string name, string country, int population, double area)
        {
            this.id = id;
            this.name = name;
            this.country = country;
            this.population = population;
            this.area = area;
        }
        public int Id
        {
            get
            {
                return id;
            }
        }
        public int Population
        {
            get
            {
                return population;
            }
        }
        public string GetCsvString()
        {
            return id + ";" + name + ";" + country + ";" + population + ";" + area;
        }
        public override string ToString()
        {
            return string.Format("ID: {0} - Name: {1} - Country: {2} - Population: {3} - Area: {4}",
id, name, country, population, area);
        }
    }
    class ListCapital
    {
```

```
        private Capital[] _items;
        private int _size;
        public ListCapital()
        {
            this._items = new Capital[16];
            this._size = 0;
        }
        public void Add(Capital newCapital)
        {
            if (_size == _items.Length)
            {
                EnsureCapasity();
            }
            this._items[this._size] = newCapital;
            this._size += 1;
        }
        public void Insert(int index, Capital newCapital)
        {
            if (_size == _items.Length)
            {
                EnsureCapasity();
            }
            for (int i = _size - 1; i >= index; i--)
            {
                _items[i + 1] = _items[i];
            }
            _items[index] = newCapital;
            _size += 1;
        }
        public bool Remove(Capital newCapital)
        {
            for (int i = 0; i < _size; i++)
            {
                if (_items[i] == newCapital)
                {
                    for (int j = i; j < _size - 1; j++)
                    {
                        _items[j] = _items[j + 1];
                    }
                    _size -= 1;
                    _items[_size] = null;
                    return true;
                }
            }
            return false;
        }
        public void RemoveAt(int index)
        {
            if (index != _size - 1)
            {
                for (int i = index; i < _size - 1; i++)
                {
                    _items[i] = _items[i + 1];
                }
            }
            _size -= 1;
            _items[_size] = null;
        }
        public void Clear()
        {
            int capacity = _items.Length;
            _items = new Capital[capacity];
            _size = 0;
        }
        public int Count
        {
            get
            {
                return _size;
            }
        }
        public int Capacity
        {
            get
```

```
            {
                return _items.Length;
            }
        }
        public Capital this[int index]
        {
            get
            {
                return _items[index];
            }
            set
            {
                _items[index] = value;
            }
        }

        public IEnumerator<Capital> GetEnumerator()
        {
            return this._items.Take(this._size).GetEnumerator();
        }

        private void EnsureCapasity()
        {
            int oldCapacity = _items.Length;
            Capital[] oldArray = _items;
            _items = new Capital[oldCapacity * 2];
            Array.Copy(oldArray, _items, oldCapacity);
        }
    }
    class Program
    {
        static ListCapital ReadAllCapitals(string filePath)
        {
            StreamReader sr = new StreamReader(filePath);
            ListCapital list = new ListCapital();
            string line = "";
            bool isFirstLine = true;
            while (true)
            {
                line = sr.ReadLine();
                if (isFirstLine)
                {
                    isFirstLine = false;
                    continue;
                }
                if (line == null)
                {
                    break;
                }
                string[] csvData = line.Split(";");
                if (csvData.Length != 5)
                {
                    throw new Exception("CSV number of column is incorrect");
                }
                else if (!int.TryParse(csvData[0], out int id))
                {
                    throw new Exception("First columnt is not integer");
                }
                else if (!int.TryParse(csvData[3], out int population))
                {
                    throw new Exception("Fourth columnt is not integer");
                }
                else if (!double.TryParse(csvData[4], out double area))
                {
                    throw new Exception("Fifth column is not double");
                }
                else
                {
                    list.Add(new Capital(id, csvData[1], csvData[2], population, area));
                }
            }
            sr.Close();
            return list;
        }
```

```
        static void WriteFirstTenElements(ListCapital list)
        {
            int length = list.Count < 10 ? list.Count : 10;
            for (int i = 0; i < length; i++)
            {
                WriteLine(list[i].ToString());
            }
        }
        static ListCapital GenerateNewList(ListCapital list1, ListCapital list2)
        {
            ListCapital resultList = new ListCapital();
            int length = list1.Count > list2.Count ? list1.Count : list2.Count;
            bool[] isElementInResultList = new bool[length];
            for (int i = 0; i < list1.Count; i++)
            {
                if (!isElementInResultList[i])
                {
                    resultList.Add(list1[i]);
                    isElementInResultList[i] = true;
                }
            }
            for (int i = 0; i < list2.Count; i++)
            {
                if (!isElementInResultList[i])
                {
                    resultList.Add(list2[i]);
                    isElementInResultList[i] = true;
                }
            }
            return resultList;
        }
        static double FindAveragePopulation(ListCapital list)
        {
            long sum = 0;
            foreach(Capital capital in list)
            {
                sum += capital.Population;
            }
            return sum / (double)list.Count;
        }
        static void RemoveItemWithLessThanAveragePopulation(double average, ListCapital list)
        {
            for (int i = 0; i < list.Count; i++)
            {
                if (list[i].Population < average)
                {
                    list.RemoveAt(i);
                    i--;
                }
            }
        }
        static void WriteAllCapitals(string filePath, ListCapital list)
        {
            StreamWriter sw = new StreamWriter(filePath);
            foreach (Capital capital in list)
            {
                sw.WriteLine(capital.GetCsvString());
            }
            sw.Close();
        }
        static void Main(string[] args)
        {
            const string file1Path = @"D:\progbase\labs\second_sem\lab1\lab1Part1\small.csv";
            const string file2Path = @"D:\progbase\labs\second_sem\lab1\lab1Part1\large.csv";
            const string file3Path = @".\output.csv";

            ListCapital list1 = ReadAllCapitals(file1Path);
            WriteLine("Size of first list: {0}", list1.Count);
            WriteFirstTenElements(list1);
            WriteLine("-------------");
            ListCapital list2 = ReadAllCapitals(file2Path);
            WriteLine("Size of second list: {0}", list2.Count);
            WriteFirstTenElements(list2);
            WriteLine("-------------");
```

```
            ListCapital list3 = GenerateNewList(list1, list2);
            double averagePopulation = FindAveragePopulation(list3);
            WriteLine("\nAverage: {0}\n", averagePopulation);
            RemoveItemWithLessThanAveragePopulation(averagePopulation, list3);
            WriteLine("-------------");
            WriteAllCapitals(file3Path, list3);
        }
    }
}
```

Приклади результатів

Частина 1. Генерація двох .csv файлів.

Перший – на 20 елементів

id	name	country	populatio	area
0	Baku	Bosnia He	927011	213,366
1	Vienna	Botswana	927562	32,949
2	Sofia	Bosnia He	415634	51,245
3	Ouagadou	Central Af	612395	118,108
4	Ouagadou	Bolivia	153833	16,863
5	Sofia	Barbados	948723	545,533
6	Sucre	Belize	862313	192,736
7	Manama	Banglades	473091	28,653
8	Sucre	Bolivia	839487	77,031
9	Copenhag	Bhutan	315651	835,393
10	Ottawa	Belgium	639897	654,697
11	Hanga Ro	Benin	882792	244,064

Другий – на 200000 елементів

id	name	country	populatio	area
0	Sucre	Belize	576383	10,149
1	New Delh	Bolivia	247913	195,523
2	Sucre	Burkina	871875	857,807
3	Copenhag	Benin	37406	23,889
4	Manama	Canada	679214	836,603
5	Sofia	Benin	150423	926,853
6	Sofia	Canada	493851	8,002
7	Helsinki	Brunei	979364	79,824
8	Ouagadou	Brazil	963723	46,18
9	West Islar	Burundi	911446	45,824
10	Asmara	Belarus	363477	125,807
11	BrasÃ-lia	Brazil	731087	441,581
12	Copenhag	Belize	251715	120,565
13	Sofia	Belgium	845227	77,332
14	Vienna	Benin	684244	453,828
15	Copenhag	Cape Verc	616253	49,426
16	Copenhag	Bolivia	90812	48,309
17	Manama	Botswana	681034	869,055
18	Sucre	Cameroor	321640	422,452
19	Baku	Benin	614293	299,166
20	BrasÃ-lia	Cape Verc	245474	688,725
21	Ouagadou	Botswana	657333	91,074
22	Ouagadou	Cape Verc	630628	13,024
23	Sucre	Belarus	32202	21,198
24	Tbilisi	Brazil	973513	613,134
25	Sucre	Central Af	3199	44,557

Частина 2. Створення .csv файлу на основі перетворених двох інших

0	Baku	Bosnia He	927011	213,366
1	Vienna	Botswana	927562	32,949
3	Ouagadou	Central Af	612395	118,108
5	Sofia	Barbados	948723	545,533
6	Sucre	Belize	862313	192,736
8	Sucre	Bolivia	839487	77,031
10	Ottawa	Belgium	639897	654,697
11	Hanga Roa	Benin	882792	244,064
15	Jerusalem	Brunei	775807	659,344
16	New Delh	Cameroor	962234	3,284
21	Ouagadou	Botswana	657333	91,074
22	Ouagadou	Cape Verc	630628	13,024
24	Tbilisi	Brazil	973513	613,134
26	Sofia	Benin	898750	141,843
28	Asmara	Cameroor	511033	206,04
31	Helsinki	Cape Verc	577269	13,146
32	Ottawa	Botswana	571530	138,149
33	Vienna	Banglades	947416	376,098
34	Canberra	Bahrain	543045	147,57
35	Jerusalem	Bahrain	637527	35,515
39	Sucre	Belize	865231	65,984
42	Helsinki	Benin	524109	189,042

Висновки

Виконавши дану лабораторну роботу було створено дві консольні утиліти. Перша генерує випадкові дані, та записує їх у файл. Шлях до файлу, розширення та кількість сутностей задається користувачем за допомогою аргументів командного рядка. Друга на основі двох .csv файлів робить списки, що потім перетворюються певним чином в один та записується в окремий .csv файл.

Компіляція всього коду відбувалася за допомогою утиліти dotnet.