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 № 6

з дисципліни “Основи програмування”

тема “Об'єкти і класи”

Виконав
студент І курсу
групи КП-02
Жученко Андрій Сергійович
(*прізвище, ім’я, по батькові*)

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

Київ 2020

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. Одне із полів має бути цілочисельним і з назвою **id** - ідентифікатор об'єкта. Додати в клас реалізацію стандартного метода ToString (див. додаток).

Створити клас (ListEntity, Entity замінити на назву типу сутності) 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 для кожного файлу окремо.

Дії (створити на кожну по функції):

- Створити новий об’єкт ListEntity і переписати в нього всі елементи з перших двох списків.
- Обрати будь-яке 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 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.
Назва функції (метода) має містити хоча б одне дієслово, що описує дію, яку вона виконує.

Тексти коду програм

Program.cs

```
using System;
using System.IO;

namespace lab1_2
{
```

```
    class Site
    {
        private int _id;
        private string _address;
        private string _topic;
        private int _numberOfVisitors;
        public Site()
        {
            _id = 0;
            _address = null;
            _topic = null;
            _numberOfVisitors = 0;
        }
        public Site(int idNumber, string link, string topicOfSite, int visitors)
        {
            _id = idNumber;
            _address = link;
            _topic = topicOfSite;
            _numberOfVisitors = visitors;
        }
        public override string ToString()
        {
            return $"ID: {this._id}, adress: {this._address}, topic of the site:
{this._topic}, avarage number of visitors in month: {this._numberOfVisitors}";
        }
        public string ConvertToCsvRaw()
        {
            return
$"{this._id},{this._address},{this._topic},{this._numberOfVisitors}";
        }
        public int GetNumberOfVisitors()
        {
            return this._numberOfVisitors;
        }
    }
    class ListSite
    {
        private Site[] _items;
        private int _size;
        public ListSite()
        {
            _items = new Site[16];
            _size = 0;
        }
```

```
        private void EnsureCapasity(int newSize)
        {
           Array.Resize(ref this._items, newSize);
        }
        private void MoveLementsLeft(int index)
        {
            for(int i = index; i < this._size - 1; i ++)
            {
                this._items[i] = this._items[i + 1];
            }
            this._size --;
        }
        private void MoveElementsRight(int index)
        {
            if(this._size == this._items.Length)
            {
                EnsureCapasity(this._items.Length * 2);
            }
            for(int i = this._size; i > index; i--)
            {
                this._items[i] = this._items[i - 1];
            }
            this._size ++;
        }
        public void Add(Site newSite)
        {
            if(this._size == this._items.Length)
            {
                EnsureCapasity(this._items.Length * 2);
            }
            this._items[this._size] = newSite;
            this._size ++;
        }
        public void Insert(int index, Site newSite)
        {
            if(index > this._size || index < 0)
            {
                throw new Exception("There is no such index in the list");
            }
            else if(index == this._size)
            {
                this.Add(newSite);
            }
            else
```

```
        {
            this.MoveElementsRight(index);
            this._items[index] = newSite;
        }
    }
    public bool Remove(Site site)
    {
        for(int i = 0; i < this._size; i++)
        {
            if(site == this._items[i])
            {
                MoveLementsLeft(i);
                return true;
            }
        }
        return false;
    }
    public void RemoveAt(int index)
    {
        if(index < 0 || index > this._size - 1)
        {
            throw new Exception("There is no such index in the list");
        }
        MoveLementsLeft(index);
    }
    public void Clear()
    {
        for(int i = 0; i < this._size; i++)
        {
            this._items[i] = null;
        }
        this._size = 0;
    }
    public int GetCount()
    {
        return this._size;
    }
    public int GetCapacity()
    {
        return this._items.Length;
    }
    public Site GetAt(int index)
    {
        if(index < 0 || index > this._size - 1)
```

```
            {
                throw new Exception("There is no such index in the list");
            }
            return this._items[index];
        }
        public void SetAt(int index, Site site)
        {
            if(index < 0 || index > this._size - 1)
            {
                throw new Exception("There is no such index in the list");
            }
            this._items[index] = site;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            string outputFile;
            int numberOfRaws;
            CheckCommandlineArgs(args, out outputFile, out numberOfRaws);
            CreateCsv(outputFile, numberOfRaws);
            ListSite sites = ReadAllSites("./sites.csv");
            ListSite sites1 = ReadAllSites("./sites1");
            Console.WriteLine("There are {0} raws in first file", sites.GetCount());
            Console.WriteLine("Here are fisrt 10 raws from first file:");
            for(int i = 0; i < 10; i++)
            {
                Console.WriteLine(sites.GetAt(i).ConvertToCsvRaw());
            }
            Console.WriteLine("There are {0} raws in second file", sites1.GetCount());
            Console.WriteLine("Here are fisrt 10 raws from second file:");
            for(int i = 0; i < 10; i++)
            {
                Console.WriteLine(sites1.GetAt(i).ConvertToCsvRaw());
            }
            ListSite newSites = UnionOfTwoLists(sites, sites1);
            int avarage = GetAvarage(newSites);
            RemoveElements(newSites, avarage);
            WriteAllEntities("./output", newSites);
        }
        static bool CheckCommandlineArgs(string[] args, out string nameOfFile, out int
numberOfRaws)
        {
```

```
            if(args.Length < 2)
            {
                throw new Exception("Not all arguments were given.");
            }
            if(args.Length > 2)
            {
                throw new Exception("Too many arguments were givem.");
            }
            int numberOfStrings;
            bool isNumberOK = int.TryParse(args[1], out numberOfStrings);
            if(isNumberOK == false)
            {
                throw new Exception("Wrong second argument. It should be an ineger");
            }
            else if(numberOfStrings < 0)
            {
                throw new Exception("Wrong number of raws.");
            }
            else
            {
                nameOfFile = args[0];
                numberOfRaws = numberOfStrings;
                return true;
            }
        }
        static void CreateCsv(string outputFile, int numberOfRaws)
        {
            StreamWriter writer = new StreamWriter(outputFile);
            writer.WriteLine("id,address,topic,number of visitors");
            Random random = new Random();
            for(int i = 0; i < numberOfRaws; i++)
            {
                Site site = new Site(i + 1, GetRandomString(), GetRandomString(),
random.Next(1000, 10000000));
                string csvRaw = site.ConvertToCsvRaw();
                writer.WriteLine(csvRaw);
            }
            writer.Close();
        }
        static string GetRandomString()
        {
            Random random = new Random();
            int length = random.Next(2, 15);
            char[] chars = new char[length];
```

```
            for(int i = 0; i < length; i++)
            {
                int randomChar = random.Next((int)'a', (int)'z' + 1);
                chars[i] = (char)randomChar;
            }
            return new string(chars);
        }
        static ListSite ReadAllSites(string filePath)
        {
            StreamReader sr = new StreamReader(filePath);
            string s = null;
            ListSite sites = new ListSite();
            int counter = 0; // this counter created in order to skip first raw in
CSV, cause first raw contains names of parameters
            while(true)
            {
                s = sr.ReadLine();
                if (s == null)
                {
                    break;
                }
                if(counter != 0)
                {
                    string[] parameters = s.Split(',');
                    CheckCsvRaw(parameters);
                    Site site = new Site(int.Parse(parameters[0]), parameters[1],
parameters[2], int.Parse(parameters[3]));
                    sites.Add(site);
                }
                counter++;
            }
            sr.Close();
            return sites;
        }
        static ListSite UnionOfTwoLists(ListSite sites, ListSite sites1)
        {
            ListSite newSites = new  ListSite();
            for(int i = 0; i < sites.GetCount(); i++)
            {
                newSites.Add(sites.GetAt(i));
            }
            for(int i = 0; i < sites1.GetCount(); i++)
            {
                newSites.Add(sites1.GetAt(i));
```

```
        }
        return newSites;
    }
    static int GetAvarage(ListSite sites)
    {
        int summ = 0;
        for(int i = 0; i < sites.GetCount(); i++)
        {
            summ = summ + sites.GetAt(i).GetNumberOfVisitors();
        }
        return summ / sites.GetCount();
    }
    static void RemoveElements(ListSite sites, int avarage)
    {
        for(int i = 0; i < sites.GetCount(); i++)
        {
            if(sites.GetAt(i).GetNumberOfVisitors() < avarage)
            {
                sites.RemoveAt(i);
            }
        }
    }
    static void WriteAllEntities(string file, ListSite sites)
    {
        StreamWriter sw = new StreamWriter(file);
        for(int i = 0; i < sites.GetCount(); i++)
        {
            sw.WriteLine(sites.GetAt(i).ConvertToCsvRaw());
        }
        sw.Close();
    }
    static void CheckCsvRaw(string[] raw)
    {
        int a;
        bool isCorrect = int.TryParse(raw[0], out a);
        int b;
        bool isCorrect1 = int.TryParse(raw[3], out b);
        if(isCorrect == false || isCorrect1 == false)
        {
            throw new Exception("Wrong data in files");
        }
        if(raw.Length != 4)
        {
            throw new Exception("Wrong data in files");
```

```
            }
        }
    }
}
```

Приклади результатів програм

Вивід перших 10 рядків з кожного списку та довжина кожного списку.

```
There are 15 raws in first file
Here are fisrt 10 raws from first file:
1,yeripyavjvn,svljddnjkpir,7142346
2,hhh,tljdl,9413337
3,snjuzq,zzfjfo,3180397
4,ovn,ocmg,1941959
5,plqmivucanwd,celiumpiorwyuf,3396990
6,vgocgaxzbxoq,ito,1997082
7,ifgja,nitneg,935484
8,juqlr,pkomxwyzrehehz,8902995
9,wzwczxhb,btavzhgnmpgar,2613256
10,mkhyqnsrkqrf,vei,2824949
There are 110000 raws in second file
Here are fisrt 10 raws from second file:
1,snhtffqiarm,ustotsbdhzb,3787206
2,xigfncd,fxydfxx,9601910
3,tmnrodlx,fxtapgxe,978696
4,ejk,bdcluhchhke,4914388
5,stklpg,zcqqyathmr,5042641
6,ugbm,esuo,4374760
7,cxttqkyyhlkit,aoah,1893421
8,hsstizcqa,ohmfjnztztunkj,9137007
9,ixjrljmzn,fwckuxwiq,7699341
10,bpbto,inljha,1118846
```

Приклад роботи при некоректних аргументах командного рядка:

```
(base) Andrews-MacBook-Pro:lab1_2 andrewzhuchenko$ dotnet run 111 ϕϕϕ
Unhandled exception. System.Exception: Wrong second argument. It should be an ineger
   at lab1_2.Program.CheckCommandlineArgs(String[] args, String& nameOfFile, Int32& numberOfRaws) in /Users/andrewzhuchenko/kpi/pb/progbase/labs/lab1_2/Prog
ram.cs:line 193
   at lab1_2.Program.Main(String[] args) in /Users/andrewzhuchenko/kpi/pb/progbase/labs/lab1_2/Program.cs:line 156
```

Приклад роботи при некоректних даних у файлах:

```
Unhandled exception. System.Exception: Wrong data in files
   at lab1_2.Program.CheckCsvRaw(String[] raw) in /Users/andrewzhuchenko/kpi/pb/progbase/labs/lab1_2/Program.cs:line 305
   at lab1_2.Program.ReadAllSites(String filePath) in /Users/andrewzhuchenko/kpi/pb/progbase/labs/lab1_2/Program.cs:line 245
   at lab1_2.Program.Main(String[] args) in /Users/andrewzhuchenko/kpi/pb/progbase/labs/lab1_2/Program.cs:line 158
```

Висновки

Протягом виконання цієї лабораторної роботи був створений та використаний клас для об’єктів з даними без логики та об’єктів динамічних колекцій елементів.

Також були згенеровані і оброблені у великої кількості дані у форматі CSV.