**Міністерство освіти і науки України**

**Національний авіаційний університет**

**Навчально-науковий інститут комп’ютерних**

**інформаційних технологій**



Лабораторна робота №3.1

з дисципліни «Об’єктивно-орієнтоване програмування»

«Дослідження роботи з файлами та проектування зв’язків між сутностями»

Виконав студент:

групи ПІ-224Б

Ляшенко Б.М.

Перевірив викладач:

Дишлевий О.П

Київ 2022

Історія контролю

|  |  |  |
| --- | --- | --- |
| **Дата** | **Опис** | **Автор** |
| 01.09.2022 | Дослідження роботи з файлами та проектування зв’язків між сутностями | Ляшенко Богдан Михайлович |

Мета – дослідити роботу з файлами з використанням потоків у мові С# при роботі з різними пов’язаними сутностями.

Завдання:

1. Дослідити структуру класів потокового введення/виведення для роботи з файлами у мові С#.

2. Спроектувати, представити у вигляді діаграми класів функціонал згідно з варіантом.

3. Розробити програму на мові C#, яка відповідає вимогам у завданні та варіанті. Для демонстрації роботи використати текстові повідомлення на консолі через операції вводу- виводу. Програма повинна показати застосування класів потокового введення/ виведення для роботи з файлами шляхом запису, читання та маніпуляцій над даними спроектованих сутностей та зв’язків між ними.

4. Діаграма(-и) та вихідний код повинні відповідати базовим принципам проектування: OOП, composition over inheritance, loose coupling – high cohesion, inversion of control (IoC).

5. Для отримання балів, що відповідають «задовільно» необхідно реалізувати завдання, відповідно варіанту. Операції роботи із файлами, вводу-виводу повинні бути в окремих класах, а не в класах бізне-сутностей.

6. Для отримання балів, що відповідають «добре» повинно бути реалізовано усе, що на «задовільно». А також:

a. В окремий проект (project) рішення (solution) виділити операції читання-запису у

файл. Тут не повинно бути ніяких маніпуляцій з даними чи ввід-вивід на консоль.

Тобто в рішенні буде мінімум 2 проекти

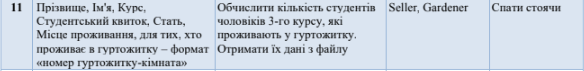
b. Операції вводу-виводу на екран – в окремому класі ConsoleMenu.

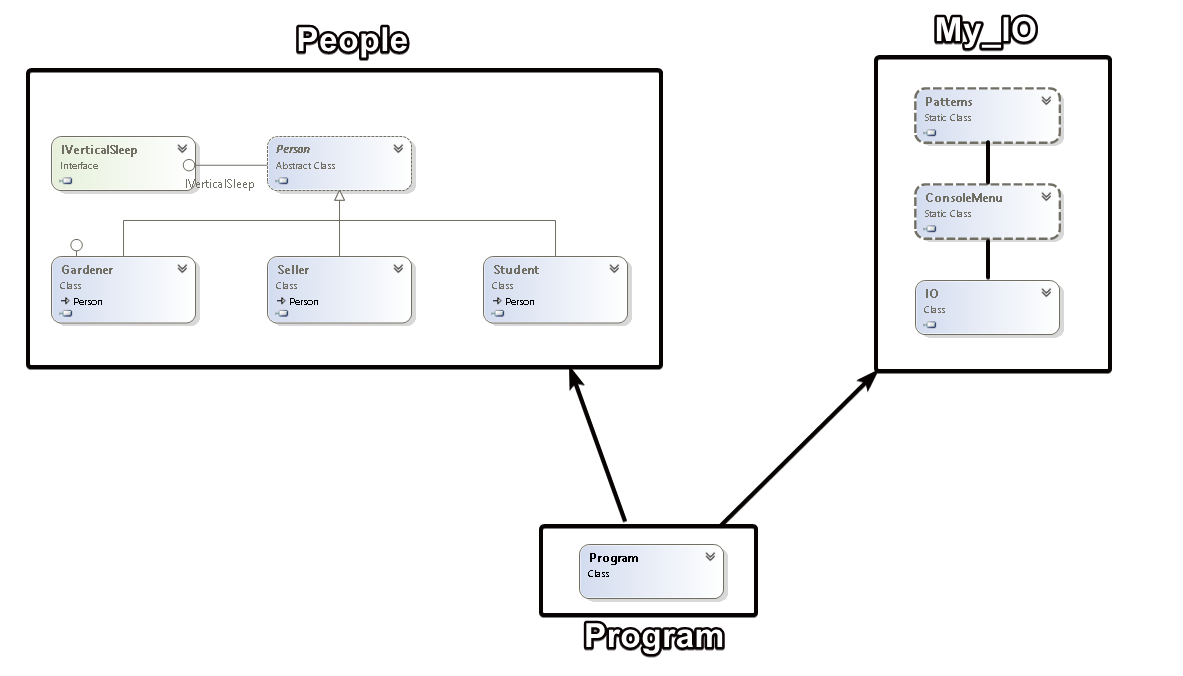
c. Також необхідно використати абстракції (абстрактні класи та інтерфейси), для

демонстрації зв’язків між сутностями у варіанті.

7. Для отримання балів, що відповідають «відмінно», спроектувати зв’язки між сутностями таким чином, щоб можна було легко додавати інші пов’язані типи (наприклад, Pupil чи Musician) без необхідності змінювати існуючі. А також – нову поведінку. Наприклад, коли операцію Play() може виконувати музикант та студент.

8. \*Завдання підвищеної складності – працювати з одинм єдиним файлом-джерелом даних, записуючи та читаючи дані різних типів та сутностей.





Хід роботи

**Program:**

using My\_IO;

using System;

using System.IO;

using System.Text.RegularExpressions;

namespace OOP\_LAB\_1

{

internal class Program

{

static void Main(string[] args)

{

ConsoleMenu.Info();

path:

Console.Write("Enter path to file where you want to write:");

string path = Console.ReadLine();

Regex regex = new Regex(@"\.txt$");

if (regex.IsMatch(path))

{

IO my = new IO();

if (!File.Exists(path))

my = new IO(path);

my.Path = path;

string input;

do

{

Console.Write("\nEnter the command:");

input = Console.ReadLine();

Console.WriteLine();

try

{

switch (input.ToLower())

{

case "/info":

ConsoleMenu.Info();

break;

case "/persons":

ConsoleMenu.Persons();

break;

case "/add":

ConsoleMenu.AddPersonToFile(my);

break;

case "/show":

ConsoleMenu.Show(my);

break;

case "/search":

ConsoleMenu.Search(my);

break;

case "/sleepvertical":

ConsoleMenu.SleepVertical(my);

break;

case "/study":

ConsoleMenu.Study(my);

break;

case "/getfiles":

ConsoleMenu.ShowAllFiles();

break;

case "/delete":

ConsoleMenu.Delete(my);

break;

case "/clear":

ConsoleMenu.ClearFile(my);

break;

case "/cls":

Console.Clear();

break;

case "/path":

goto path;

case "/end":

break;

default:

Console.WriteLine("Ther is no such a command");

break;

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

} while (input.ToLower() != "/end");

}

else

{

Console.WriteLine("Wrong file path! Try enter again.");

goto path;

}

}

}

**People:**

Person:

using System.Text.RegularExpressions;

namespace People

{

public abstract class Person : IVerticalSleep

{

string firstName;

string lastName;

string sex;

string residence;

Regex validName = new Regex(@"^[A-Za-z ]+$");

System.ArgumentException nameIsNotValid = new System.ArgumentException("Text is not valid!!!");

public Person()

{

}

public Person(string firstName, string lastName, string sex, string residence)

{

FirstName = firstName;

LastName = lastName;

Sex = sex;

Residence = residence;

}

public string FirstName

{

get

{

return firstName;

}

set

{

if (validName.IsMatch(value))

firstName = value;

else

throw nameIsNotValid;

}

}

public string LastName

{

get

{

return lastName;

}

set

{

if (validName.IsMatch(value))

lastName = value;

else

throw nameIsNotValid;

}

}

public string Sex

{

get { return sex; }

set

{

if (validName.IsMatch(value))

sex = value;

else

throw nameIsNotValid;

}

}

public string Residence

{

get { return residence; }

set

{

if (validName.IsMatch(value))

residence = value;

else

throw nameIsNotValid;

}

}

public virtual string SleepVertical()

{

string res = $"{Regex.Replace(this.GetType().ToString(), @"\w+.(?<name>\w+)", @"${name}")}" +

$" with name {FirstName} {LastName} is sleeping vertical in {Residence}";

return res;

}

}

}

IVerticalSleep:

namespace People

{

internal interface IVerticalSleep //<T> where T : Person

{

string SleepVertical();

}

}

Student:

using System;

using System.Text.RegularExpressions;

namespace People

{

public class Student : Person

{

string residence;

string studentId;

string course;

public Student()

{

}

public Student(string firstName, string lastName, string sex, bool isLivingInDorm, string residence,

string studentId, string course)

{

FirstName = firstName;

LastName = lastName;

Sex = sex;

StudentId = studentId;

IsLivingInDorm = isLivingInDorm;

Residence = residence;

Course = course;

}

public bool IsLivingInDorm { get; set; }

public string StudentId

{

get

{

return studentId;

}

set

{

Regex regex = new Regex(@"^[A-Z]{2}\s\d{8}$");

if (regex.IsMatch(value))

studentId = value;

else

throw new ArgumentException("Wrong Student Id!");

}

}

new public string Residence

{

get { return residence; }

set

{

if (IsLivingInDorm)

{

Regex regexDorm = new Regex(@"^\d{1,2}-\d{3}$");

if (regexDorm.IsMatch(value))

residence = value;

else

throw new ArgumentException("Wrong Format of Residence!");

}

else

residence = value;

}

}

public string Course

{

get { return course; }

set

{

Regex regex = new Regex("[1-6]");

if (regex.IsMatch(value))

course = value;

else

throw new ArgumentException("Wrong Course!");

}

}

public string Study()

{

string res = $"Student with name: {FirstName} {LastName} is studying in {Course} year";

return res;

}

public override string SleepVertical()

{

string res = $"{Regex.Replace(this.GetType().ToString(), @"\w+.(?<name>\w+)", @"${name}")}" +

$" with name: {FirstName} {LastName} is sleeping vertical in {Residence}";

return res;

}

public override string ToString()

{

string res = $"Student {FirstName}{LastName}\n" +

$"\"{{ \"firstname\": \"{FirstName}\",\n" +

$"\"lastname\": \"{LastName}\",\n" +

$"\"sex\": \"{Sex}\",\n" +

$"\"residence\": \"{Residence}\",\n" +

$"\"course\": \"{Course}\",\n" +

$"\"studentId\": \"{StudentId}\"}};";

return res;

}

}

}

Seller:

namespace People

{

public class Seller : Person

{

public Seller()

{

}

public Seller(string firstName, string lastName, string sex, string residence, string product)

: base(firstName, lastName, sex, residence)

{

Product = product;

}

public string Product { get; set; }

public string Sell()

{

string res = $"Seller with name: {FirstName} {LastName} is now selling {Product}";

return res;

}

public override string ToString()

{

string res = $"Seller {FirstName}{LastName}\n" +

$"\"{{ \"firstname\": \"{FirstName}\",\n" +

$"\"lastname\": \"{LastName}\",\n" +

$"\"sex\": \"{Sex}\",\n" +

$"\"residence\": \"{Residence}\",\n" +

$"\"product\": \"{Product}\"}};";

return res;

}

}

}

Gardener:

namespace People

{

public class Gardener : Person, IVerticalSleep

{

public Gardener()

{

}

public Gardener(string firstName, string lastName, string sex, string residence, string employer)

: base(firstName, lastName, sex, residence)

{

Employer = employer;

}

public string Employer { get; set; }

public string CutTheLawn()

{

string res = $"Gardener with name: {FirstName} {LastName} is now cutting the lawn in {Employer} residence";

return res;

}

public override string ToString()

{

string res = $"Gardener {FirstName}{LastName}\n" +

$"\"{{ \"firstname\": \"{FirstName}\",\n" +

$"\"lastname\": \"{LastName}\",\n" +

$"\"sex\": \"{Sex}\",\n" +

$"\"residence\": \"{Residence}\",\n" +

$"\"employer\": \"{Employer}\"}};";

return res;

}

}

}

**My\_IO:**

Patterns:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Text.RegularExpressions;

namespace My\_IO

{

public static class Patterns

{

static string paternStudent = "Student [A-Za-z]+\\s+" +

"{ \"firstname\": \"(?<firstname>[A-Za-z]+)\",\\s+" +

"\"lastname\": \"(?<lastname>[A-Za-z]+)\",\\s+" +

"\"sex\": \"(?<sex>[A-Za-z]+)\",\\s+" +

"\"residence\": \"(?<residence>[A-Za-z ]+)\",\\s+" +

"\"course\": \"(?<course>[1-6]{1})\",\\s+" +

"\"studentId\": \"(?<studentId>[A-Za-z]{2}\\s\\d{8})\"};";

static string paternStudentDorm = "Student [A-Za-z]+\\s+" +

"{ \"firstname\": \"(?<firstname>[A-Za-z]+)\",\\s+" +

"\"lastname\": \"(?<lastname>[A-Za-z]+)\",\\s+" +

"\"sex\": \"(?<sex>[A-Za-z]+)\",\\s+" +

"\"residence\": \"(?<residence>\\d{1,2}-\\d{3})\",\\s+" +

"\"course\": \"(?<course>[1-6]{1})\",\\s+" +

"\"studentId\": \"(?<studentId>[A-Za-z]{2}\\s\\d{8})\"};";

static string paternGardener = "Gardener [A-Za-z]+\\s+" +

"{ \"firstname\": \"(?<firstname>[A-Za-z]+)\",\\s+" +

"\"lastname\": \"(?<lastname>[A-Za-z]+)\",\\s+" +

"\"sex\": \"(?<sex>[A-Za-z]+)\",\\s+" +

"\"residence\": \"(?<residence>[A-Za-z ]+)\",\\s+" +

"\"employer\": \"(?<employer>[A-Za-z ]+\\s\*[A-Za-z]+)\"};";

static string paternSeller = "Seller [A-Za-z]+\\s+" +

"{ \"firstname\": \"(?<firstname>[A-Za-z]+)\",\\s+" +

"\"lastname\": \"(?<lastname>[A-Za-z]+)\",\\s+" +

"\"sex\": \"(?<sex>[A-Za-z]+)\",\\s+" +

"\"residence\": \"(?<residence>[A-Za-z ]+)\",\\s+" +

"\"product\": \"(?<product>[A-Za-z ]+)\"};";

static string paternPerson = "\\w+ [A-Za-z]+\\s+" +

"{ \"firstname\": \"(?<firstname>[A-Za-z]+)\",\\s+" +

"\"lastname\": \"(?<lastname>[A-Za-z]+)\",\\s+" +

"\"sex\": \"(?<sex>[A-Za-z]+)\",\\s+" +

"\"residence\": ";

static string patern3D = "Student [A-Za-z]+\\s+" +

"{ \"firstname\": \"(?<firstname>[A-Za-z]+)\",\\s+" +

"\"lastname\": \"(?<lastname>[A-Za-z]+)\",\\s+" +

"\"sex\": \"(?<sex>[A-Za-z]+)\",\\s+" +

"\"residence\": \"(?<residence>\\d{1,2}-\\d{3})\",\\s+" +

"\"course\": \"(?<course>[3])\",\\s+" +

"\"studentId\": \"(?<studentId>[A-Za-z]{2}\\s\\d{8})\"};";

public static Regex regex3D = new Regex(patern3D);

public static Regex regexSt = new Regex(paternStudent);

public static Regex regexG = new Regex(paternGardener);

public static Regex regexSe = new Regex(paternSeller);

public static Regex regexStD = new Regex(paternStudentDorm);

public static Regex regexPer = new Regex(paternPerson);

}

}

IO:

using People;

using System;

using System.IO;

using System.Text;

using System.Text.RegularExpressions;

namespace My\_IO

{

public class IO

{

public IO()

{

} //+

public IO(string path)

{

Path = path;

FileStream stream = File.Create(Path);

stream.Close();

}//+

public int Count()

{

int count = Patterns.regexPer.Matches(DisplayAllFromFile()).Count;

return count;

} //+

public string Path { get; set; } //+

public void WriteDownPerson<T>(T person) where T : Person

{

StringBuilder res = new StringBuilder();

res.Append($"{Regex.Replace(person.GetType().ToString(), @"\w+.(?<name>\w+)", @"${name}")}" +

$" {person.FirstName}{person.LastName}\n");

res.Append($"{{ \"firstname\": \"{person.FirstName}\",\n" +

$"\"lastname\": \"{person.LastName}\",\n" +

$"\"sex\": \"{person.Sex}\",\n" +

$"\"residence\": \"{person.Residence}");

if (person is Student)

{

Student student = person as Student;

res.Append($"{student.Residence}\",\n");

res.Append($"\"course\": \"{student.Course}\",\n");

res.Append($"\"studentId\": \"{student.StudentId}\"}};");

}

else if (person is Seller)

{

Seller seller = person as Seller;

res.Append($"\",\n\"product\": \"{seller.Product}\"}};");

}

else if (person is Gardener)

{

Gardener gardener = person as Gardener;

res.Append($"\",\n\"employer\": \"{gardener.Employer}\"}};");

}

res.AppendLine(Environment.NewLine);

File.AppendAllText(Path, res.ToString());

} //+

public Person[] ReadAllFromFile()

{

if (Count() > 0)

{

Person[] persons = new Person[Count()];

int index = 0;

for (Match m = Patterns.regexSt.Match(DisplayAllFromFile()); m.Success; m = m.NextMatch())

{

persons[index] = new Student(m.Groups["firstname"].ToString(),

m.Groups["lastname"].ToString(), m.Groups["sex"].ToString(),

false, m.Groups["residence"].ToString(), m.Groups["studentId"].ToString(), m.Groups["course"].ToString());

index++;

}

for (Match m = Patterns.regexStD.Match(DisplayAllFromFile()); m.Success; m = m.NextMatch())

{

persons[index] = new Student(m.Groups["firstname"].ToString(),

m.Groups["lastname"].ToString(), m.Groups["sex"].ToString(),

true, m.Groups["residence"].ToString(), m.Groups["studentId"].ToString(), m.Groups["course"].ToString());

index++;

}

for (Match m = Patterns.regexG.Match(DisplayAllFromFile()); m.Success; m = m.NextMatch())

{

persons[index] = new Gardener(m.Groups["firstname"].ToString(),

m.Groups["lastname"].ToString(), m.Groups["sex"].ToString(),

m.Groups["residence"].ToString(), m.Groups["employer"].ToString());

index++;

}

for (Match m = Patterns.regexSe.Match(DisplayAllFromFile()); m.Success; m = m.NextMatch())

{

persons[index] = new Seller(m.Groups["firstname"].ToString(),

m.Groups["lastname"].ToString(), m.Groups["sex"].ToString(),

m.Groups["residence"].ToString(), m.Groups["product"].ToString());

index++;

}

return persons;

}

else

return null;

} //+

public Person[] Search()

{

if (SearchString().count > 0)

{

Person[] persons = new Person[SearchString().count];

int index = 0;

for (Match m = Patterns.regex3D.Match(DisplayAllFromFile()); m.Success; m = m.NextMatch())

{

persons[index] = new Student(m.Groups["firstname"].ToString(),

m.Groups["lastname"].ToString(), m.Groups["sex"].ToString(),

true, m.Groups["residence"].ToString(), m.Groups["studentId"].ToString(), m.Groups["course"].ToString());

index++;

}

return persons;

}

else

return null;

} //+

public void ClearFile()

{

File.WriteAllText(Path, "");

}

//+

#region String

public string DisplayAllFromFile()

{

string res = File.ReadAllText(Path);

return res;

}

public (string list, int count) SearchString()

{

StringBuilder stringBuilder = new StringBuilder();

int count = 0;

for (Match m = Patterns.regex3D.Match(DisplayAllFromFile()); m.Success; m = m.NextMatch())

{

stringBuilder.Append(m.Value + Environment.NewLine + Environment.NewLine);

count++;

}

return (stringBuilder.ToString(), count);

}

#endregion

}

}

ConsoleMenu:

using People;

using System;

using System.Reflection;

using System.Text.RegularExpressions;

using System.IO;

namespace My\_IO

{

public static class ConsoleMenu

{

public static void Info()

{

Console.WriteLine(

"All commands:\n" +

" /add - add new person to file;\n" +

" /delete - delete selected person from file;\n" +

" /clear - clears file;\n" +

" /getfiles - show all txt files in directory;\n\n" +

" /info - show list of commands;\n" +

" /persons - show list of persons, who you can add to file;\n" +

" /search - show students who live in dorm and study in the 3rd year;\n" +

" /show - show whole file; \n\n" +

" /sleepvertical - all persons in file start to sleep vertical;\n" +

" /study - all student starts to study;\n\n" +

" /end - end program.");

} //+

public static void Persons()

{

//Assembly asm = Assembly.Load("People");

foreach (Type t in Assembly.Load("People").GetTypes()) // Type T in asm.GetTypes()

{

Type person = typeof(Person);

//Type person = Type.GetType("People.Person");

//Type person = Type.GetType("People.Person, People", false, true);

if (t.IsSubclassOf(person))

Console.WriteLine(t.Name);

}

} //+

public static void AddPersonToFile(IO my)

{

Console.WriteLine("Enter who you want to add to file:");

Persons();

Console.Write("Person: ");

string person = Console.ReadLine();

switch (person.ToLower())

{

case ("student"):

Student student = new Student();

FillInfoAboutPerson(student, my);

break;

case ("seller"):

Seller seller = new Seller();

FillInfoAboutPerson(seller, my);

break;

case ("gardener"):

Gardener gardener = new Gardener();

FillInfoAboutPerson(gardener, my);

break;

default:

Console.WriteLine("Wrong type of person");

break;

}

} //+

static void FillInfoAboutPerson<T>(T person, IO my) where T : Person

{

string name = Regex.Replace(person.GetType().ToString(), @"\w+.(?<name>\w+)", @"${name}");

Console.Write($"Enter First Name of a {name}: ");

person.FirstName = Console.ReadLine();

Console.Write($"Enter Last Name of a {name}: ");

person.LastName = Console.ReadLine();

Console.Write($"Enter Sex of a {name}: ");

person.Sex = Console.ReadLine();

if (person is Student)

{

Student student = person as Student;

Console.Write($"Enter Course of a {name}: ");

student.Course = Console.ReadLine();

Console.Write($"Enter Student Id of a {name}: ");

student.StudentId = Console.ReadLine();

Console.Write("Does student live in dorm: ");

string dorm = Console.ReadLine();

if (dorm.ToLower() == "yes")

student.IsLivingInDorm = true;

else

student.IsLivingInDorm = false;

Console.Write($"Enter Residence of a {name}: ");

student.Residence = Console.ReadLine();

}

else

{

if (person is Seller)

{

Seller seller = person as Seller;

Console.Write($"Enter Product of a {name}: ");

seller.Product = Console.ReadLine();

}

else if (person is Gardener)

{

Gardener gardener = person as Gardener;

Console.Write($"Enter Empolyer of a {name}: ");

gardener.Employer = Console.ReadLine();

}

Console.Write($"Enter Residence of a {name}: ");

person.Residence = Console.ReadLine();

}

my.WriteDownPerson(person);

} //+

public static void Search(IO my)

{

var search = my.SearchString();

Console.WriteLine($"There are {search.count} students who live in dorm and study in 3rd course\n");

Person[] persons = my.Search();

if (persons == null)

return;

for(int i = 0; i < search.count; i++)

{

Console.WriteLine($"{i+1}." + persons[i] + Environment.NewLine);

}

} //+

public static void Show(IO my)

{

Person[] persons = my.ReadAllFromFile();

if (persons == null)

throw new FileNotFoundException("File is empty!!!"); ;

for (int i = 0; i < persons.Length; i++)

{

Console.WriteLine($"{i+1}." + persons[i] + Environment.NewLine);

}

} //+

public static void ClearFile(IO my)

{

my.ClearFile();

Console.WriteLine("File was cleared");

} //+

public static void ShowAllFiles()

{

Console.Write("Enter Path to Directory: ");

string path = Console.ReadLine();

DirectoryInfo directory = new DirectoryInfo(path);

foreach (var file in directory.GetFiles("\*.txt"))

{

Console.WriteLine(file.Name);

}

} //+

public static void Delete(IO my)

{

Show(my);

Console.Write("\nEnter index of person who you want to deleter from file:");

int index = int.Parse(Console.ReadLine());

Person[] persons = my.ReadAllFromFile();

my.ClearFile();

Array.Copy(persons, index, persons, index-1,

persons.Length - index);

persons[persons.Length - 1] = null;

if (persons.Length <= 1)

{

Show(my);

return;

}

for(int i = 0; i < persons.Length -1; i++)

{

my.WriteDownPerson(persons[i]);

}

} //+

public static void SleepVertical(IO my)

{

Person[] people = my.ReadAllFromFile();

if (people == null)

{

Console.WriteLine("File is empty!!!");

return;

}

foreach (Person person in people)

{

Console.WriteLine(person.SleepVertical());

}

}

public static void Study(IO my)

{

Person[] people = my.ReadAllFromFile();

if (people == null)

{

Console.WriteLine("File is empty!!!");

return;

}

foreach (Student student in people)

{

Console.WriteLine(student.Study());

}

}

}

}

Висновок:

Я дослідив роботу з файлами з використанням потоків у мові С# при роботі з різними пов’язаними сутностями.