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 № 3

з дисципліни “Основи програмування”
тема “Модульне програмування і контракти”

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

Виконати розділення коду програми на модулі.
Використати контракти (інтерфейси) для розділення коду клієнта та реалізації та забезпечення змінності реалізації.

Постановка завдання

Завдання

Програма дозволяє користувачу виконувати через консоль операції над двома множинами цілих чисел A і B. На початку роботи програми обидві множини порожні.

Модуль командного інтерфейсу

Реалізувати **модуль командного інтерфейсу користувача**, через який користувач задає текстову команду у консолі і отримує результат її виконання або опис помилки:

- Команда `**{set} add {value}**` додає значення {value} у множину {set} і виводить результат додавання (true/false).
  Приклад: `**a add 13**`, `**b add -200**`
- Команда `**{set} contains {value}**` перевіряє чи значення {value} є у множині {set} і виводить результат (true/false).
  Приклад: `**a contains 13**`, `**b contains -200**`
- Команда `**{set} remove {value}**` видаляє значення {value} з множини {set} і виводить результат видалення (true/false).
  Приклад: `**a remove 13**`, `**b remove -200**`

- Команда `**{set} clear**` очищує множину {set} (робить її порожньою).
  Приклад: `**a clear**`, `**b clear**`
- Команда `**{set} log**` виводить числа з множини {set}.
  Приклад: `**a log**`, `**b log**`
- Команда `**{set} count**` виводить кількість елементів у множині {set}.
  Приклад: `**a count**`, `**b count**`
- Команда `**{set} read {file}**` читає з файлу {file} унікальні числа у множину {set} (спосіб запису чисел у файлі за варіантом з **додатку В**).
  Файли можна попередньо генерувати випадковим чином або записати вручну.
  Приклади: `**a read ./file1.txt**`, `**b read ./b.txt**`
- Команда `**{set} write {file}**` записує у файл {file} числа з множини {set} спосіб запису чисел у файлі за варіантом з **додатку В**).
  Приклади: `**a write ./out.txt**`, `**b write ./x.txt**`
- Команда `**{operation}**` виконує над множинами A і B одну із операцій за варіантом (див. **додаток А**) і виводить результат. Множина A - перший операнд, множина B - другий операнд.
  Приклад: `**IntersectWith**`, `**Overlaps**`

Користувач вводить команди у циклі. Можна перервати цикл спеціальною командою або порожньою командою.
Виникнення будь-якої помилки перехоплювати і виводити, користувач продовжує працювати з програмою.

Модуль логування

Весь вивід результатів роботи програми та повідомлень про помилки виконувати через обраний **модуль логування**.

Реалізувати два модулі логування для різних способів логування повідомлень: перший - у консоль, другий за варіантом (див. **додаток С**).

Модуль логування обирається через перший аргумент командного рядка. Приклади запуску програми: `**dotnet run console**` або `**dotnet run csv ./file.csv**` За замовчуванням (якщо не задано аргументу командного рядка) використовувати модуль логування у консоль.

Текст коду програми

Program.cs

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

namespace lab3
{
    class Program
    {
        static void GetHelp()
        {
            string[] commands = new string[] { "{set} add {value}", "{set} contains {value}", "{set} remove {value}", "{set} cle
ar",
                "{set} log", "{set} count", "{set} read {filePath}", "{set} write {filePath}", "{set} setEquals", "{set} symmetr
icExceptWith" };
            WriteLine("List of commands: ");
            foreach (string command in commands)
            {
                WriteLine(command);
            }
        }
        struct CommandParams
        {
            public string command;
            public string set;
            public int intValue;
            public string filePath;
        }
        static CommandParams GetParams(string input)
        {
            string[] subcommands = input.Split(" ");
            CommandParams commandParams = new CommandParams();
            if (subcommands.Length == 2)
            {
                if (subcommands[1] == "clear")
                {
                    commandParams.command = "clear";
                }
                else if (subcommands[1] == "log")
                {
                    commandParams.command = "log";
                }
                else if (subcommands[1] == "count")
                {
                    commandParams.command = "count";
                }
                else if (subcommands[1] == "setEquals")
                {
                    commandParams.command = "setEquals";
                }
                else if (subcommands[1] == "symmetricExceptWith")
                {
                    commandParams.command = "symmetricExceptWith";
                }
                else
                {
                    throw new Exception("Unknown command");
                }
            }
            else if (subcommands.Length == 3)
            {
                if (subcommands[1] == "add")
                {
                    commandParams.command = "add";
```

```
                if (!int.TryParse(subcommands[2], out commandParams.intValue))
                {
                    throw new Exception("Value is not integer");
                }
            }
            else if (subcommands[1] == "contains")
            {
                commandParams.command = "contains";
                if (!int.TryParse(subcommands[2], out commandParams.intValue))
                {
                    throw new Exception("Value is not integer");
                }
            }
            else if (subcommands[1] == "remove")
            {
                commandParams.command = "remove";
                if (!int.TryParse(subcommands[2], out commandParams.intValue))
                {
                    throw new Exception("Value is not integer");
                }
            }
            else if (subcommands[1] == "read")
            {
                commandParams.command = "read";
                if (File.Exists(subcommands[2]))
                {
                    commandParams.filePath = subcommands[2];
                }
                else
                {
                    throw new Exception("File not found");
                }
            }
            else if (subcommands[1] == "write")
            {
                commandParams.command = "write";
                commandParams.filePath = subcommands[2];
            }
            else
            {
                throw new Exception("Unknown command");
            }
        }
        else
        {
            throw new Exception("Wrong command length");
        }

        if (subcommands[0] == "a")
        {
            commandParams.set = "a";
        }
        else if (subcommands[0] == "b")
        {
            commandParams.set = "b";
        }
        else
        {
            throw new Exception("Wrong set");
        }
        return commandParams;
    }
    static bool Add(ISetInt set, int value)
    {
        return set.Add(value);
    }
    static bool Contains(ISetInt set, int value)
    {
        return set.Contains(value);
    }
    static bool Remove(ISetInt set, int value)
```

```
        {
            return set.Remove(value);
        }
        static void Clear(ISetInt set)
        {
            set.Clear();
        }
        static string Log(ISetInt set)
        {
            int[] array = new int[set.Count];
            set.CopyTo(array);
            string result = "";
            for (int i = 0; i < array.Length; i++)
            {
                result += array[i];
                result += i != array.Length - 1 ? "," : ".";
            }
            return result;
        }
        static int Count(ISetInt set)
        {
            return set.Count;
        }
        static void ReadSet(string filePath, ISetInt set)
        {
            StreamReader sr = new StreamReader(filePath);
            string line;
            while (true)
            {
                line = sr.ReadLine();
                if (line == null)
                {
                    break;
                }
                if (!int.TryParse(line, out int value))
                {
                    continue;
                }
                set.Add(value);
            }
            sr.Close();
        }
        static void WriteSet(string filePath, ISetInt set)
        {
            int[] array = new int[set.Count];
            set.CopyTo(array);
            StreamWriter sw = new StreamWriter(filePath);
            for (int i = 0; i < array.Length; i++)
            {
                sw.WriteLine(array[i]);
            }
            sw.Close();
        }
        static bool SetEquals(ISetInt set1, ISetInt set2)
        {
            return set1.SetEquals(set2);
        }
        static void SymmetricExceptWith(ISetInt set1, ISetInt set2)
        {
            set1.SymmetricExceptWith(set2);
        }
        static void ProcessSets(ILogger logger)
        {
            SetInt setA = new SetInt();
            SetInt setB = new SetInt();
            bool exit = false;
            while (!exit)
            {
                Write("> ");
                string input = ReadLine();
                if (input == "exit" || input == "")
```

```
            {
                exit = true;
                continue;
            }
            else if (input == "help")
            {
                GetHelp();
                continue;
            }
            CommandParams commandParams;
            try
            {
                commandParams = GetParams(input);
            }
            catch (Exception ex)
            {
                logger.LogError($"Error: {ex.Message}");
                continue;
            }
            ISetInt thisSet = commandParams.set == "a" ? setA : setB;
            ISetInt otherSet = commandParams.set == "a" ? setB : setA;
            switch (commandParams.command)
            {
                case "add":
                    if (Add(thisSet, commandParams.intValue))
                        logger.Log($"Value was added to set {commandParams.set}");
                    else
                        logger.LogError($"Value was not added to set {commandParams.set}");
                    break;
                case "contains":
                    if (Contains(thisSet, commandParams.intValue))
                        logger.Log($"Value contains in set {commandParams.set}");
                    else
                        logger.LogError($"Value doesn't contain in set {commandParams.set}");
                    break;
                case "remove":
                    if (Remove(thisSet, commandParams.intValue))
                        logger.Log($"Value was removed from set {commandParams.set}");
                    else
                       logger.LogError($"Value was not removed from set {commandParams.set}");
                    break;
                case "clear":
                    Clear(thisSet);
                    logger.Log($"Set {commandParams.set} was cleared");
                    break;
                case "log":
                    logger.Log($"Set {commandParams.set} values: {Log(thisSet)}");
                    break;
                case "count":
                    logger.Log($"Number of elements in set {commandParams.set}: {Count(thisSet)}");
                    break;
                case "read":
                    ReadSet(commandParams.filePath, thisSet);
                    logger.Log($"Set {commandParams.set} was updated");
                    break;
                case "write":
                    WriteSet(commandParams.filePath, thisSet);
                    logger.Log($"Set {commandParams.set} was written to {commandParams.filePath}");
                    break;
                case "setEquals":
                    logger.Log(SetEquals(thisSet, otherSet) ? "Sets are equal" : "Sets are not equal");
                    break;
                case "symmetricExceptWith":
                    SymmetricExceptWith(thisSet, otherSet);
                    logger.Log($"Set {commandParams.set} was updated");
                    break;
            }
        }
    }
    struct Options
    {
```

```
            public string loggerType;
            public string filePath;
            public string errors;
        }
        static Options ParseOptions(string[] args)
        {
            if (args.Length == 0 || (args.Length == 1 && args[0] == "console"))
            {
                return new Options() { loggerType = "console" };
            }
            else if (args.Length == 1)
            {
                return new Options() { loggerType = "csv", filePath = args[0] };
            }
            else
            {
                return new Options() { errors = "Unknown logger type" };
            }
        }
        static void Main(string[] args)
        {
            Options options = ParseOptions(args);
            if (options.errors != null)
            {
                WriteLine(options.errors);
                return;
            }

            if (options.loggerType == "console")
            {
                ConsoleLogger logger = new ConsoleLogger();
                ProcessSets(logger);
            }
            else
            {
                CsvFileLogger logger = new CsvFileLogger(options.filePath);
                ProcessSets(logger);
                logger.Close();
            }
        }
    }
}
```

ILogger.cs

```
using System;

namespace lab3
{
    interface ILogger
    {
        void Log(string message);
        void LogError(string errorMessage);
    }
}
```

ISetInt.cs

```
using System;

namespace lab3
{
    interface ISetInt
    {
        int Count { get; }

        bool Add(int value);
        bool Remove(int value);
```

```
        bool Contains(int value);
        void Clear();

        void CopyTo(int[] array);

        void SymmetricExceptWith(ISetInt other);
        bool SetEquals(ISetInt other);
    }
}
```

ConsoleLogger.cs

```
using System;

namespace lab3
{
    class ConsoleLogger : ILogger
    {
        public void Log(string message)
        {
            Console.WriteLine(message);
        }

        public void LogError(string errorMessage)
        {
            Console.Error.WriteLine(errorMessage);
        }
    }
}
```

CsvFileLogger.cs

```
using System;
using System.IO;

namespace lab3
{
    class CsvFileLogger : ILogger
    {
        private StreamWriter sw;
        public CsvFileLogger(string fileString)
        {
            this.sw = new StreamWriter(fileString);
        }
        public void Log(string message)
        {
            sw.WriteLine(DateTime.Now.ToString("o") + "," + "LOG" + "," + message);
        }

        public void LogError(string errorMessage)
        {
            sw.WriteLine(DateTime.Now.ToString("o") + "," + "ERROR" + "," + errorMessage);
        }
        public void Close()
        {
            sw.Close();
        }
    }
}
```

SetInt.cs

```
using System;

namespace lab3
{
    class SetInt : ISetInt
    {
```

```
        private int size;
        private int[] array;

        public SetInt()
        {
            this.size = 0;
            this.array = new int[16];
        }

        public int Count
        {
            get { return this.size; }
        }

        public bool Add(int value)
        {
            if (this.size == this.array.Length)
            {
                EnsureCapacity();
            }
            if (Contains(value))
            {
                return false;
            }
            int i = size - 1;
            while (i >= 0 && this.array[i] > value)
            {
                this.array[i + 1] = this.array[i];
                i--;
            }
            this.array[i + 1] = value;
            this.size++;
            return true;
        }

        public void Clear()
        {
            this.size = 0;
            this.array = new int[this.array.Length];
        }

        public bool Contains(int value)
        {
            int index = GetIndex(value);
            return index != -1;
        }

        public void CopyTo(int[] array)
        {
            if (array.Length != this.size)
            {
                throw new ArgumentException();
            }
            Array.Copy(this.array, array, this.size);
        }

        public bool Remove(int value)
        {
            int index = GetIndex(value);
            if (index == -1)
            {
                return false;
            }
            for (int i = index; i < this.size - 1; i++)
            {
                this.array[i] = this.array[i + 1];
            }
            this.array[this.size] = 0;
            this.size--;
            return true;
        }
```

```
        public bool SetEquals(ISetInt other)
        {
            int[] otherArray = new int[this.size];
            try
            {
                other.CopyTo(otherArray);
            }
            catch
            {
                return false;
            }
            for (int i = 0; i < this.size; i++)
            {
                if (this.array[i] != otherArray[i])
                {
                    return false;
                }
            }
            return true;
        }

        public void SymmetricExceptWith(ISetInt other)
        {
            int[] otherArray = new int[other.Count];
            other.CopyTo(otherArray);
            for (int i = 0; i < otherArray.Length; i++)
            {
                if (Contains(otherArray[i]))
                {
                    Remove(otherArray[i]);
                }
                else
                {
                    Add(otherArray[i]);
                }
            }
        }
        private void EnsureCapacity()
        {
            Array.Resize<int>(ref this.array, this.array.Length * 2);
        }
        private int GetIndex(int value)
        {
            int low = 0;
            int high = this.size - 1;
            while (low <= high)
            {
                int mid = (low + high) / 2;
                if (this.array[mid] == value)
                {
                    return mid;
                }
                else if (value < this.array[mid])
                {
                    high = mid - 1;
                }
                else
                {
                    low = mid + 1;
                }
            }
            return -1;
        }
    }
}
```

Приклади результатів

Приклад 1. Робота програми з ConsoleLogger

Console

```
D:\progbase\labs\second_sem\lab3>dotnet run console
> a log
Set a values:
> b log
Set b values:
> a add 3
Value was added to set a
> a add 3
Value was not added to set a
> a add 4
Value was added to set a
> a add 1
Value was added to set a
> a log
Set a values: 1,3,4.
> a write out.txt
Set a was written to out.txt
> b read out.txt
Set b was updated
> b log
Set b values: 1,3,4.
> b setEquals
Sets are equal
```

out.txt

```
1
3
4
```

Приклад 2. Робота з програми з CsvFileLogger

Console

```
D:\progbase\labs\second_sem\lab3>dotnet run ./log.csv
> a log
> b log
> a add 3
> b add 3
> a add 4
> a symmetricExceptWith
> a log
> b setEquals
> a add 3
> b add 4
> a setEquals
> a remove 4
> a log
> b clear
> b log
> a contains 0
```

log.csv

```
2021-03-28T21:54:06.0564587+03:00,LOG,Set a values:
2021-03-28T21:54:08.2780054+03:00,LOG,Set b values:
```

```
2021-03-28T21:54:18.8323455+03:00,LOG,Value was added to set a
2021-03-28T21:54:22.8743303+03:00,LOG,Value was added to set b
2021-03-28T21:54:32.5195430+03:00,LOG,Value was added to set a
2021-03-28T21:54:50.7161158+03:00,LOG,Set a was updated
2021-03-28T21:55:01.2964779+03:00,LOG,Set a values: 4.
2021-03-28T21:55:21.6269128+03:00,LOG,Sets are not equal
2021-03-28T21:55:30.8942635+03:00,LOG,Value was added to set a
2021-03-28T21:55:33.7580625+03:00,LOG,Value was added to set b
2021-03-28T21:55:41.6555029+03:00,LOG,Sets are equal
2021-03-28T21:55:52.7172193+03:00,LOG,Value was removed from set a
2021-03-28T21:55:56.5706892+03:00,LOG,Set a values: 3.
2021-03-28T21:56:00.7836656+03:00,LOG,Set b was cleared
2021-03-28T21:56:04.3278717+03:00,LOG,Set b values:
2021-03-28T21:56:16.8639821+03:00,ERROR,Value doesn't contain in set a
```

Приклад 3. Оброблення помилок при неправильному написанні аргументів командного рядка.

Console

```
D:\progbase\labs\second_sem\lab3>dotnet run incorrect input
Unknown logger type
```

Висновки

Виконавши дану лабораторну роботу було розроблено два окремих та незалежних модуля: консольного інтерфейсу та роботи з АТД множина. Програма дозволяє за допомогою користувацького вводу вносити зміни до структури даних.

Компіляція коду відбувалася за допомогою утиліти dotnet.