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	Перевірив
студент І курсу	“____” “_____” 20___ р.
групи КП-01	викладач
Беліцький Олександр Сергійович (*прізвище, ім’я, по батькові*)	Гадиняк Руслан Анатолійович (*прізвище, ім’я, по батькові*)

варіант № 3

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

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} (кожне число з нового рядка). 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} (кожне число з нового рядка).
  Приклади: `a write ./out.txt`, `b write ./x.txt`
- Команда `Overlaps` перевіряє чи поточна множина та інша множина мають спільні елементиі виводить результат. Множина A - перший операнд, множина B - другий операнд.

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:

CsvFileLogger2	Писати повідомлення і помилки у два файли у форматі CSV. CSV таблиця кожного файлу має містити два стовпці: **timestamp,message**. У першому стовпці записувати рядок часу у форматі ISO 8601, у другому саме повідомлення. Якщо CSV неекрановане, рекомендується перед записом повідомлення видаляти символи, що потребують екранування. Шляхи до файлів задавати через конструктор.

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

Аналіз вимог і проектування

Кроки виконання завдання:

- реалізувати модуль командного інтерфейсу із функціями обробки всіх команд та виводом у консоль
- додати контракт множини і реалізувати її класом, створити змінні з множинами та прив'язати обробку команд із викликами методів множин
- створити функції для зчитування-запису множин у файли
- додати контракт логування і реалізувати клас ConsoleLogger
- замінити у модулі командного інтерфейсу прямий вивід у консоль на вивід через методи модуля логування у консоль.
- реалізувати другий клас логування
- додати обробку аргументів командного рядка, за якою визначати і підставляти потрібний модуль логування

Вимоги до коду:

- Кожен тип (клас, інтерфейс) розмістити у окремому файлі з кодом. Назва файлу має відповідати назві типу і мати розширення `**.cs**`.
- Модифікувати задані у вказівках контракти заборонено
- Весь код має відповідати вимогам іменування

Модуль командного інтерфейсу користувача

Створити у модулі функцію `ProcessSets` (головна функція модуля командного інтерфейсу):
Викликати дану функцію з головної функції програми і передати у неї один із об'єктів логування відповідно до аргументу командного рядка та варіанту завдання.

Створити у цій функції дві змінні типу `ISetInt`, що представляють множини A і B.
Запустити цикл обробки команд. Вивід результату команд та помилки писати через об'єкт логування, що був переданий у параметр `ILogger logger`.

Винести код, що обробляє кожну з заданих команд у окрему функцію. `ProcessSets` не має бути великою.

Контракт множини цілих чисел

Дано базовий інтерфейс множини цілих чисел:

```
// множина унікальних цілих чисел, порядок зберігання не важливий
interface ISetInt
{
   int GetCount();  // отримати кількість елементів множини (можна використати властивість)

   bool Add(int value);  // додати число у множину, якщо такого числа там не було - повернути true, інакше - false
   bool Remove(int value);  // видалити число з множини, якщо такого числа там не було - повернути false, інакше - true
   bool Contains(int value);  // перевірити чи число є у множині, якщо такого числа там нема - повернути false, інакше - true
   void Clear();  // зробити множину порожньою

   void CopyTo(int[] array);  // скопіювати всі числа з множини у масив, що переданий через параметр (порядок запису чисел не важливий). Якщо довжина переданого масиву замала, викидати System.ArgumentException
}
```

Додати до інтерфейсу `ISetInt` методи за варіантом:

`bool Overlaps(ISetInt other)`	Перевіряє чи поточна множина та інша множина мають спільні елементи

Створити клас, що реалізує інтерфейс `ISetInt`.

Вимоги до реалізації:

- Методи Add та Remove реалізувати так, щоби вони мали середню алгоритмічну складність не гіршу за O(n).
- Метод Contains реалізувати так, щоби він мав середню алгоритмічну складність кращу за O(n).
- Метод (або властивість) GetCount має бути O(1).

Контракт логування

Контракт `ILogger` - інтерфейс логування повідомлень роботи програми:

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

Створити два класи, що реалізують даний інтерфейс.
Перший клас - **ConsoleLogger**, другий **CsvFileLogger2**.

Створити у головній функції програми змінну типу `ILogger`, значення якої задавати об’єктом одного із реалізованих класів логування в залежності від аргументу командного рядка.

Якщо спосіб логування передбачає використання назви (назв) файлів чи інші значення, задавати їх у аргументах командного рядка, зчитувати і передавати у конструктор відповідного об’єкта логування.
Приклади запуску програми: `**dotnet run console**` або `**dotnet run csv ./message.csv ./error.csv**`

Додаткові функції

Реалізувати функцію, що читає числа із текстового файлу у множину:

```
static ISetInt ReadSet(string filePath);
```

Реалізувати функцію, що записує числа з множини у текстовий файл:

```
static void WriteSet(string filePath, ISetInt set);
```

Тексти коду програм

Program.cs

```
using System;
using System.IO;

class Program
{
    static void Main(string[] args)
    {
        // Alpha();
        if ((args.Length == 1 && args[0] == "console") || args.Length == 0)
        {
            ILogger logger = new ConsoleLogger();

            ProccessSets(logger);
        }
        else if (args.Length == 3 && args[0] == "csv")
        {
            if (File.Exists($"{args[1]}"))
            {
                if (File.Exists($"{args[2]}"))
                {
                    ILogger logger = new CsvFileLogger2(args[1], args[2]);

                    ProccessSets(logger);
                }
                else
                {
                    throw new FileNotFoundException("Specified file does not found");
                }
            }
            else
            {
                throw new FileNotFoundException("Specified file does not found");
            }
        }
        else
        {
            throw new FormatException("Check command format to use logger");
        }
    }

    static void PrintCommands()
    {
        Console.WriteLine(@"
Available commands to work with the program:

/c o m m a n d/   /f o r m a t/                /e x e c u t i o n/
+-------------------------------------------------------------------------------------------------------
----------+
add             - {set} add {value}         -  adds the value {value} to the set {set}
contains        - {set} contains {value}    -  checks if the value {value} is in the set {set}
remove          - {set} remove {value}      -  removes the value {value} from the set {set}
clear           - {set} clear               -  clears the set {set}
log             - {set} log                 -  outputs numbers from the set {set}
count           - {set} count               -  displays the number of elements in the set {set}
read            - {set} read {file}         -  reads unique {plural} numbers from {file}
write           - {set} write {file}        -  writes to file {file} numbers from the set {set}
Overlaps        - Overlaps                  -  checks whether the current set and another set have common
elements
exit            - exit                      -  exit the program
+-------------------------------------------------------------------------------------------------------
----------+
            ");
    }
```

```
    static void ProccessSets(ILogger logger)
    {
        ISetInt a = new SetInt();
        ISetInt b = new SetInt();
        while (true)
        {
           // PrintCommands();
            Console.ForegroundColor = ConsoleColor.Green;
            Console.Write("Enter command: ");
            Console.ForegroundColor = ConsoleColor.Cyan;
            string str = Console.ReadLine();
            string[] command = str.Split(' ');

            Console.ForegroundColor = ConsoleColor.Yellow;
            if (str == "exit")
            {
                return;
            }
            else if (str.Contains(" add "))
            {
                ConsoleInterface.AddProcess(a, b, command, logger);
            }
            else if (str.Contains(" contains "))
            {
                ConsoleInterface.ContainsProccess(a, b, command, logger);
            }
            else if (str.Contains(" remove "))
            {
                ConsoleInterface.RemoveProccess(a, b, command, logger);
            }
            else if (str.Contains(" clear"))
            {
                ConsoleInterface.ClearProccess(a, b, command, logger);
            }
            else if (str.Contains(" log"))
            {
                ConsoleInterface.LogProccess(a, b, command, logger);
            }
            else if (str.Contains(" count"))
            {
                ConsoleInterface.CountProccess(a, b, command, logger);
            }
            else if (str.Contains(" read "))
            {
                ConsoleInterface.ReadProccess(a, b, command, logger);
            }
            else if (str.Contains(" write "))
            {
                ConsoleInterface.WriteProccess(a, b, command, logger);
            }
            else if (str.Contains("Overlaps"))
            {
                if (command.Length != 1)
                {
                    logger.LogError("Error: check correctness of command");
                }
                else
                {
                    logger.Log("Tip: Do sets have common numbers: " + a.Overlaps(b).ToString());
                }
            }
            else{
                logger.LogError("Error: unknown command");
            }
            Console.ResetColor();
        }
    }
}
```

ConsoleInterface.cs

```
using System;
using System.IO;
using static System.Console;
static class ConsoleInterface
{
    public static void AddProcess(ISetInt aSet, ISetInt bSet, string[] command, ILogger logger)
    {
        ISetInt set = new SetInt();
        if (command.Length != 3)
        {
            logger.LogError("Error: check correctness of command");
        }
        else
        {
            if (CheckSet(command))
            {
                set = command[0] == "a" || command[0] == "A" ? aSet : bSet;
                if (command[1] == "add")
                {
                    if (int.TryParse(command[2], out int result))
                    {
                        logger.Log("Tip: is the number added: " + set.Add(result));
                    }
                    else
                    {
                        logger.LogError("Error: check correctness of value. It must be integer number");
                    }
                }
                else
                {
                    logger.LogError("Error: check correctness of command format");
                }
            }
            else
            {
                logger.LogError("Error: check correctness of name of set. There only can be A or B ");
            }
        }
    }

    static bool CheckSet(string[] command)
    {
        string set = command[0].ToLower();
        return ((set == "a") || (set == "b"));
    }

    public static void ContainsProccess(ISetInt aSet, ISetInt bSet, string[] command, ILogger logger)
    {
        ISetInt set = new SetInt();
        if (command.Length != 3)
        {
            logger.LogError("Error: check correctness of command");
        }
        else
        {
            if (CheckSet(command))
            {
                set = command[0] == "a" || command[0] == "A" ? aSet : bSet;
                if (command[1] == "contains")
                {
                    if (int.TryParse(command[2], out int result))
                    {
                        logger.Log("Tip: does the set contains the number: " + set.Contains(result));
                    }
                    else
                    {
```

```
                    logger.LogError("Error: check correctness of value. It must be integer number");
                }
            }
            else
            {
                logger.LogError("Error: check correctness of command format");
            }
        }
        else
        {
            logger.LogError("Error: check correctness of name of set. There only can be A or B ");
        }
    }
}

public static void RemoveProccess(ISetInt aSet, ISetInt bSet, string[] command, ILogger logger)
{
    ISetInt set = new SetInt();
    if (command.Length != 3)
    {
        logger.LogError("Error: check correctness of command");
    }
    else
    {
        if (CheckSet(command))
        {
            set = command[0] == "a" || command[0] == "A" ? aSet : bSet;
            if (command[1] == "remove")
            {
                if (int.TryParse(command[2], out int result))
                {
                    logger.Log("Tip: is the number removed: " + set.Remove(result));
                }
                else
                {
                    logger.LogError("Error: check correctness of value. It must be integer number");
                }
            }
            else
            {
                logger.LogError("Error: check correctness of command format");
            }
        }
        else
        {
            logger.LogError("Error: check correctness of name of set. There only can be A or B ");
        }
    }
}

public static void ClearProccess(ISetInt aSet, ISetInt bSet, string[] command, ILogger logger)
{
    ISetInt set = new SetInt();
    if (command.Length != 2)
    {
        logger.LogError("Error: check correctness of command");
    }
    else
    {
        if (CheckSet(command))
        {
            set = command[0] == "a" || command[0] == "A" ? aSet : bSet;
            if (command[1] == "clear")
            {
                set.Clear();
                logger.Log($"Tip: set {command[0]} was cleared");
            }
            else
            {
                logger.LogError("Error: check correctness of command format");
            }
```

```
            }
            else
            {
                logger.LogError("Error: check correctness of name of set. There only can be A or B ");
            }
        }
    }

    public static void LogProccess(ISetInt aSet, ISetInt bSet, string[] command, ILogger logger)
    {
        ISetInt set = new SetInt();
        if (command.Length != 2)
        {
            logger.LogError("Error: check correctness of command");
        }
        else
        {
            if (CheckSet(command))
            {
                set = command[0] == "a" || command[0] == "A" ? aSet : bSet;
                if (command[1] == "log")
                {
                    PrintSet(set, command);
                }
                else
                {
                    logger.LogError("Error: check correctness of command format");
                }
            }
            else
            {
                logger.LogError("Error: check correctness of name of set. There only can be A or B ");
            }
        }
    }

    static void PrintSet(ISetInt set, string[] command)
    {
        int[] arr = new int[set.GetCount];
        set.CopyTo(arr);
        Write($"Set {command[0]} is:");
        foreach (var i in arr)
        {
            Write(" " + i);
        }
        WriteLine();
    }

    public static void CountProccess(ISetInt aSet, ISetInt bSet, string[] command, ILogger logger)
    {
        ISetInt set = new SetInt();
        if (command.Length != 2)
        {
            logger.LogError("Error: check correctness of command");
        }
        else
        {
            if (CheckSet(command))
            {
                set = command[0] == "a" || command[0] == "A" ? aSet : bSet;
                if (command[1] == "count")
                {
                    logger.Log($"Tip: number of elements of set {command[0]}: " + set.GetCount);
                }
                else
                {
                    logger.LogError("Error: check correctness of command format");
                }
            }
            else
            {
```

```
            logger.LogError("Error: check correctness of name of set. There only can be A or B ");
        }
    }
}

public static void ReadProccess(ISetInt aSet, ISetInt bSet, string[] command, ILogger logger)
{
    ISetInt set = new SetInt();
    if (command.Length != 3)
    {
        logger.LogError("Error: check correctness of command");
    }
    else
    {
        if (CheckSet(command))
        {
            set = command[0] == "a" || command[0] == "A" ? aSet : bSet;
            if (command[1] == "read")
            {
                if (File.Exists($"{command[2]}"))
                {
                    ReadSet(command[2], logger, set);
                    logger.Log("Tip: file was readed");
                }
                else
                {
                    logger.LogError("Error: the specified file does not exist");
                }
            }
            else
            {
                logger.LogError("Error: check correctness of command format");
            }
        }
        else
        {
            logger.LogError("Error: check correctness of name of set. There only can be A or B ");
        }
    }
}

static bool ReadSet(string filePath, ILogger logger, ISetInt set)
{
    var sr = new StreamReader(filePath);
    string s = "";
    while (true)
    {
        s = sr.ReadLine();
        if (s == null)
        {
            break;
        }
        if (int.TryParse(s, out int num))
        {
            set.Add(num);
        }
        else
        {
            logger.LogError("Error: check data correctness of reading file");
        }
    }
    return true;
}

public static void WriteProccess(ISetInt aSet, ISetInt bSet, string[] command, ILogger logger)
{
    ISetInt set = new SetInt();
    if (command.Length != 3)
    {
        logger.LogError("Error: check correctness of command");
    }
```

```
        else
        {
            if (CheckSet(command))
            {
                set = command[0] == "a" || command[0] == "A" ? aSet : bSet;
                if (command[1] == "write")
                {
                    if (File.Exists($"{command[2]}"))
                    {
                        WriteSet(command[2], set);
                        logger.Log("Tip: set was wrote in the file");
                    }
                    else
                    {
                        logger.LogError("Error: the specified file does not exist");
                    }
                }
                else
                {
                    logger.LogError("Error: check correctness of command format");
                }
            }
            else
            {
                logger.LogError("Error: check correctness of name of set. There only can be A or B ");
            }
        }
    }

    static void WriteSet(string filePath, ISetInt set)
    {
        File.WriteAllText($"{filePath}", String.Empty);
        int[] setArray = new int[set.GetCount];
        set.CopyTo(setArray);
        var sw = new StreamWriter(filePath, true);
        for (int i = 0; i < setArray.Length; i++)
        {
            sw.WriteLine(setArray[i]);
        }
        sw.Close();
    }
}
```

ConsoleLogger.cs

```
using System;

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

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

CsvFileLogger2.cs

```
using System;
using System.IO;

class CsvFileLogger2 : ILogger
{
    private string _messageCSV;
    private string _errorCSV;
    private string _dateTime;

    public CsvFileLogger2(string messageCSV, string errorCSV)
    {
        this._messageCSV = messageCSV;
        this._errorCSV = errorCSV;
        this._dateTime = "";
    }

    public void Log(string message)
    {
        _dateTime = DateTime.UtcNow.ToString();

        string str = _dateTime+ ',' + message;
        var sw = new StreamWriter(_messageCSV, true);
        sw.WriteLine(str);
        sw.Close();

    }

    public void LogError(string errorMessage)
    {
        _dateTime = DateTime.UtcNow.ToString();

        string str = _dateTime+ ',' + errorMessage;
        var sw = new StreamWriter(_errorCSV, true);
        sw.WriteLine(str);
        sw.Close();
    }
}
```

ILogger.cs

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

ISetInt.cs

```
// множина унікальних цілих чисел, порядок зберігання не важливий
interface ISetInt
{
int GetCount { get; }           // отримати кількість елементів множини (можна використати властивість)
bool Add(int value);            // додати число у множину, якщо такого числа там не було - повернути
true, інакше - false
bool Remove(int value);         // видалити число з множини, якщо такого числа там не було - повернути
false, інакше - true
bool Contains(int value);       // перевірити чи число є у множині, якщо такого числа там нема -
повернути false, інакше - true
void Clear();                   // зробити множину порожньою
void CopyTo(int[] array);       // скопіювати всі числа з множини у масив, що переданий
                                // через параметр (порядок запису чисел не важливий). Якщо довжина
переданого масиву
                                // замала, викидати System.ArgumentException
bool Overlaps(ISetInt other);   // Перевіряє чи поточна множина та інша множина мають спільні елементи
}
```

SetInt.cs

```
 using System;

class SetInt : ISetInt
{
    private int[] _items;
    private int _size;
    public SetInt()
    {
        _items = new int[0];
        _size = 0;
    }

    public bool Add(int value)
    {
        if (Contains(value))
        {
            return false;
        }

        if (this._size == this._items.Length)
        {
            Expand();
        }
        this._items[this._size] = value;
        this._size += 1;
        _items = this.InsertionSort(_items);
        return true;
    }
    private void Expand()
    {
        int oldCapacity = this._items.Length;
        int[] oldArray = this._items;
        this._items = new int[oldCapacity + 1];
        System.Array.Copy(oldArray, this._items, oldCapacity);
    }
    private int[] InsertionSort(int[] inputArray)
    {
        for (int i = 0; i < inputArray.Length - 1; i++)
        {
            for (int j = i + 1; j > 0; j--)
```

```
            {
                if (inputArray[j - 1] > inputArray[j])
                {
                    int temp = inputArray[j - 1];
                    inputArray[j - 1] = inputArray[j];
                    inputArray[j] = temp;
                }
            }
        }
        return inputArray;
    }

    public void Clear()
    {
        _size = 0;
    }

    public bool Contains(int value)
    {
        int index = FindIndex(value);
        return index >= 0;
    }

    private int FindIndex(int value)
    {
        for (int i = 0; i < _size; i++)
        {
            if (_items[i] == value)
            {
                return i;
            }
        }

        return -1;
    }

    public void CopyTo(int[] array)
    {
        if (array.Length < _size)
        {
            throw new System.ArgumentException("Error: Given array does not have enough capacity. ");
        }
        for (int i = 0; i < _size; i++)
        {
            array[i] = _items[i];
        }
    }

    public int GetCount
    {
        get
        {
            return _size;
        }
    }

    public bool Overlaps(ISetInt other)
    {
        int[] array = new int[other.GetCount];
        other.CopyTo(array);
        for (int i = 0; i < _size; i++)
        {
            for(int j = 0; j < array.Length; j++)
            {
                if(array[j] == _items[i])
                {
                    return true;
                }
            }
        }
        return false;
```

```
    }

    public bool Remove(int value)
    {
        int index = FindIndex(value);
        if (index == -1)
        {
            return false;
        }

        for (int i = index; i < _size - 1; i++)
        {
            _items[i] = _items[i + 1];
        }
        _size--;
        return true;
    }
}
```

Приклади результатів

Множини можна задавати незалежно від регістру.

- неправильні вхідні дані
- правильні вхідні дані
- початок групи команд
- команда не передбачена для виконання
- повторне використання команди

Для наглядності у цьому блоці буде використано **ConsoleLogger:**

*{set} **add** {integer value}*

{unknown set} add {integer value}

```
Enter command: c add 5
Error: check correctness of name of set. There only can be A or B
```

{set} add {value} {value}

```
Enter command: a add smth smth
Error: check correctness of command
```

{set} add {incorrect value}

```
Enter command: a add smth
Error: check correctness of value. It must be integer number
```

{set} add {integer value}

```
Enter command: a add 5
Tip: is the number added: True
```

{set} add {integer value} - repeatedly

```
Enter command: a add 5
Tip: is the number added: False
```

*{set} **contains** {integer value}*

*{unknown set} **contains** {integer value}*

```
Enter command: c contains 5
Error: check correctness of name of set. There only can be A or B
```

*{set} **contains** {value} {value}*

```
Enter command: a contains smth smth
Error: check correctness of command
```

*{set} **contains** {incorrect value}*

```
Enter command: a contains smth
Error: check correctness of value. It must be integer number
```

*{set} **contains** {integer value}*

```
Enter command: a contains 5
Tip: does the set contains the number: True
```

{set} ***remove*** *{integer value}*

{unknown set} ***remove*** *{integer value}*

```
Enter command: c remove 5
Error: check correctness of name of set. There only can be A or B
```

{set} ***remove*** *{value} {value}*

```
Enter command: a remove smth smth
Error: check correctness of command
```

{set} ***remove*** *{incorrect value}*

```
Enter command: a remove smth
Error: check correctness of value. It must be integer number
```

{set} ***remove*** *{integer value}*

```
Enter command: a remove 5
Tip: is the number removed: True
```

{set} ***remove*** *{integer value} - repeatedly*

```
Enter command: a remove 5
Tip: is the number removed: False
```

{set} ***clear***

{unknown set} ***clear***

```
Enter command: c clear
Error: check correctness of name of set. There only can be A or B
```

{set} ***clear*** *{value}*

```
Enter command: a clear smth
Error: check correctness of command
```

{set} ***clear***

```
Enter command: a clear
Tip: set a was cleared
```

{set} ***clear*** *- repeatedly*

```
Enter command: a clear
Tip: set a was cleared
```

{set} ***log*** **- попередньо додамо елементи `-2`,`0`,`4`**

{unknown set} ***log***

```
Enter command: c log
Error: check correctness of name of set. There only can be A or B
```

{set} ***log*** *{value}*

```
Enter command: a log smth
Error: check correctness of command
```

{set} ***log***

```
Enter command: a log
Set a is: -2 0 4
```

{set} ***count***

{unknown set} ***count***

```
Enter command: c count
Error: check correctness of name of set. There only can be A or B
```

{set} ***count*** *{value}*

```
Enter command: a count smth
Error: check correctness of command
```

{set} ***count***

```
Enter command: a count
Tip: number of elements of set a: 3
```

{set} ***read*** *{integer value}*

{unknown set} ***read*** *{filepath}*

```
Enter command: c read ./data.txt
Error: check correctness of name of set. There only can be A or B
```

{set} ***read*** *{filepath} {value}*

```
Enter command: b read ./data.txt smth
Error: check correctness of command
```

{set} ***read*** *{incorrect filepath }*

```
Enter command: b read ./notexist.txt
Error: the specified file does not exist
```

{set} ***read*** *{filepath}*

```
Enter command: b read ./data.txt
Tip: file was readed
```

Set after reading:

```
Enter command: b log
Set b is: -2 1 4 5
```

{set} ***read*** *{filepath} - repeatedly(add only unique elements)*

```
Enter command: b read ./data.txt
Tip: file was readed
```

Set after rereading:

```
Enter command: b log
Set b is: -2 1 4 5
```

*{set} **write** {integer value}*

*{unknown set} **write** {filepath}*

```
Enter command: c write ./data.txt
Error: check correctness of name of set. There only can be A or B
```

*{set} **write** {filepath} {value}*

```
Enter command: b write ./data.txt smth
Error: check correctness of command
```

*{set} **write** {incorrect filepath }*

```
Enter command: b write ./notexist.txt
Error: the specified file does not exist
```

*{set} **write** {filepath}*

```
Enter command: b write ./data.txt
Tip: set was wrote in the file
```

Set after writting:

data.txt

```
1  -2
2  1
3  4
4  5
```

*{set} **read** {filepath} - repeatedly(rewritting all file)*

Попередньо змінимо множину b, очистивши та додавши елементи `4`,`7`,`9`

```
Enter command: b write ./data.txt
Tip: set was wrote in the file
```

Set after rewritting:

data.txt

```
1  4
2  7
3  9
```

Overlaps

***Overlaps** {value}*

```
Enter command: Overlaps smth
Error: check correctness of command
```

Overlaps

Sets before execution:

```
Set a is: -2 1 4 5
Set b is: 4 7 9
Enter command: Overlaps
Tip: Do sets have common numbers: True
```

Remove `4` from b set.

Sets before execution:

```
Set a is: -2 1 4 5
Set b is:  7 9
Enter command: Overlaps
Tip: Do sets have common numbers: False
```

exit

exit *{value}*

```
Enter command: exit ss
Error: unknown command
```

exit

```
Enter command: exit
Ending proccessing...
```

unknowncommand

```
Enter command: unknowncommand
Error: unknown command
```

Висновки

Виконавши дану лабораторну роботу був виконаний розподіл коду на модулі. Модуль - функціонально закінчений фрагмент програми, оформлений у вигляді окремого файлу з вихідним кодом.

Також в даній лабораторній роботі були використані інтерфейси. Інтерфейс являє собою референс тип, який може визначати деякий функціонал - набір методів і властивостей без реалізації. Потім цей функціонал реалізують класи і структури, які застосовують дані інтерфейси. Інтерфейс може задати загальну ознаку для різнорідних об'єктів, а це відкриває величезні можливості по частині гнучкості коду.

Також у даній лабораторній роботі було ралізовано абстрактний тип даних Множина. Дані типу Множини дозволяють зберігати обмежене число значень певного типу без певного порядку. Повторення значень - неприпустимо. У цій лабораторній роботі було розроблено реалізацію Множини для цілих чисел.

Компіляція всього коду відбувалася за допомогою утиліти dotnet.