

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(ла)  студент(ка) I 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

Виконати розділення коду програми на модулі.

Використати контракти (інтерфейси) для розділення коду клієнта та реалізації та забезпечення змінності реалізації.

Постановка завданняЗавдання

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 B**). 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 B**).   
  Приклади: `**a write ./out.txt**`, `**b write ./x.txt**`
* Команда `**{operation}**` виконує над множинами A i B одну із операцій за варіантом (див. **додаток A**) і виводить результат. Множина А - 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 C**).

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.

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](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/identifier-names)

## Модуль командного інтерфейсу користувача

“Інтерфейс користувача” у даному випадку означає будь-які засоби для взаємодії користувача з програмою і не має нічого спільного з interface.

Модулем командного інтерфейсу можна зробити клас Program або створити новий клас (наприклад, CommandUserInterface), у якому розміщувати статичні функції, у яких через консоль відбувається ввід даних користувачем та вивід даних програми у відповідь.

Створити у модулі функцію ProcessSets (головна функція модуля командного інтерфейсу):

|  |
| --- |
| static void ProcessSets(ILogger logger)  {  throw new NotImplementedException();  } |

Викликати дану функцію з головної функції програми (Main) 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 (див. **додаток A**).

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 - відповідно до варіанту (див. **додаток С**).

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: `**dotnet run csv ./file.csv**`, `**dotnet run plain ./file1.txt ./errors.txt**`.

## 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); |

Спосіб зберігання чисел у файлах за варіантом із **додатку B**.

## Додаток А. Варіант додаткових методів множини

**4** - номер у списку групи.

*Таблиця 1. Варіант першого метода*

|  |  |
| --- | --- |
| **n % 4** | **Методи (див. опис у таблиці 3)** |
| 0 | SetEquals |

|  |  |
| --- | --- |
| bool SetEquals(ISetInt other) | Перевіряє чи поточна та інша множини містять однаковий набір елементів. |

*Таблиця 2. Варіант другого метода*

|  |  |
| --- | --- |
| **n % 3** | **Методи (див. опис у таблиці 3)** |
| 1 | UnionWith |

|  |  |
| --- | --- |
| void UnionWith(ISetInt other) | Модифікує поточну множину так, що вона містить всі елементи з цієї та іншої множини. |

## Додаток B. Варіант способу запису чисел у файлах

*Таблиця 4. Варіант способу запису чисел у файлах*

|  |  |
| --- | --- |
| **n % 3** | **Спосіб** |
| 1 | Числа у файлі записані через коми |

## Додаток C. Варіант класу логування

*Таблиця 5. Варіант класу логування*

|  |  |
| --- | --- |
| **n % 5** | **Клас (див. опис у таблиці 6)** |
| 4 | SqliteFileLogger |

|  |  |
| --- | --- |
| SqliteLogger | Писати повідомлення у таблицю **logs** БД SQLite. Таблиця має містити три стовпці: **timestamp,type,message**.  У першому стовпці записувати рядок часу у форматі [ISO 8601](https://en.wikipedia.org/wiki/ISO_8601), у другому тип повідомлення (LOG|ERROR), у третьому саме повідомлення.  Шлях до файлу задавати через конструктор. |

**Тексти коду програм**

|  |
| --- |
| progbase/Labs/lab3\_2/Program.cs |
| using System;  using static System.Console;  using System.IO;  namespace lab3\_2  {  class Program  {  static void Main(string[] args)  {  if(args.Length == 0 || (args[0] == "console" && args.Length == 1))  {  ILogger logger = new ConsoleLogger();  ProcessSet(logger);  }  else if(args.Length == 2 && args[0] == "sql")  {  string dataBaseFile = args[1];  ILogger logger = new SQLLogger(dataBaseFile);  ProcessSet(logger);  }  else  {  WriteLine("Wrong command line arguments.");  }  }  static void ProcessSet(ILogger logger)  {  WriteLine("Set is based on bool array, so user must set range of possible values himself.");  WriteLine("Please, input minimal and maximal values that can be putted into set.");  WriteLine("Input min for set a:");  string minA = ReadLine();  WriteLine("Input max for set a:");  string maxA = ReadLine();  WriteLine("Input min for set b:");  string minB = ReadLine();  WriteLine("Input max for set b:");  string maxB = ReadLine();  if(CheckInput(minA, maxA, minB, maxB) == false)  {  WriteLine("Wrong input! Only integer can be inputted. Besides, minimal values shoud be lower than maximal.");  }  else  {  int minimalA = int.Parse(minA);  int maximalA = int.Parse(maxA);  int minimalB = int.Parse(minB);  int maximalB = int.Parse(maxB);  IsetInt a = new ArraySetInt(minimalA, maximalA);  IsetInt b = new ArraySetInt(minimalB, maximalB);  while(true)  {  WriteLine("Input your command:");  string[] command = ReadLine().Split(" ");  if(command.Length == 1)  {  if(command[0] == "exit")  {  break;  }  else if(command[0] == "union")  {  ProcessCommandUnion(a, b, logger);  }  else if(command[0] == "equals")  {  ProcessCommandEqual(a, b, logger);  }  else  {  logger.LogError("Wrong command.");  }  }  else if(command.Length == 3)  {  if(command[1] == "add")  {  ProcessCommandAdd(command, ref a, ref b, minimalA, maximalA, minimalB, maximalB, logger);  }  else if(command[1] == "write")  {  ProcessCommandWrite(command, a, b, logger);  }  else if(command[1] == "contains")  {  ProcessCommandContains(command, a, b, minimalA, maximalA, minimalB, maximalB, logger);  }  else if(command[1] == "read")  {  ProcessCommandRead(command, a, b, minimalA, maximalA, minimalB, maximalB, logger);  }  else if(command[1] == "remove")  {  ProcessCommandRemove(command, a, b, minimalA, maximalA, minimalB, maximalB, logger);  }  else  {  logger.LogError("Wrong command.");  }  }  else if(command.Length == 2)  {  if(command[1] == "log")  {  ProcessCommandLog(command, a, b, logger);  }  else if(command[1] == "count")  {  ProcessCommandCount(command, a, b, logger);  }  else if(command[1] == "clear")  {  ProcessCommandClear(command, a, b, logger);  }  else  {  logger.LogError("Wrong command.");  }  }  else  {  logger.LogError("Wrong command.");  }  }  }  }  static bool CheckInput(string minA, string maxA, string minB, string maxB)  {  int minimalA;  int maximalA;  int minimalB;  int maximalB;  bool isInt1 = int.TryParse(minA, out minimalA);  bool isInt2 = int.TryParse(maxA, out maximalA);  bool isInt3 = int.TryParse(minB, out minimalB);  bool isInt4 = int.TryParse(maxB, out maximalB);  if(isInt1 == false || isInt2 == false || isInt3 == false || isInt4 == false)  {  return false;  }  if(minimalA >= maximalA || minimalB >= maximalB)  {  return false;  }  return true;  }  static bool CkeckSetValue(string value)  {  int a;  bool isOK = int.TryParse(value, out a);  return isOK;  }  static void WriteSet(string filePath, IsetInt set)  {  StreamWriter sw = new StreamWriter(filePath);  int[] arrayOfSet = new int[set.GetCount()];  set.CopyTo(arrayOfSet);  for(int i = 0; i < arrayOfSet.Length; i++)  {  sw.Write("{0},", arrayOfSet[i]);  }  sw.Close();  }  static void ProcessCommandUnion(IsetInt a, IsetInt b, ILogger logger)  {  a.UnionWith(b);  int[] array = new int[a.GetCount()];  a.CopyTo(array);  string set = String.Join(',', array);  logger.Log(set);  }  static void ProcessCommandEqual(IsetInt a, IsetInt b, ILogger logger)  {  bool Equals = a.SetEquals(b);  logger.Log(Equals.ToString());  }  static void ProcessCommandAdd(string[] command, ref IsetInt setA, ref IsetInt setB, int minA, int maxA, int minB, int maxB, ILogger logger)  {  if(CkeckSetValue(command[2]) == false)  {  logger.LogError("Only integers can be putted into set.");  return;  }  int newValue = int.Parse(command[2]);  if(command[0] == "a")  {  if(newValue < minA || newValue > maxA)  {  logger.LogError("Value is not in range of possible values.");  return;  }  bool add = setA.Add(newValue);  logger.Log(add.ToString());  }  else if(command[0] == "b")  {  if(newValue < minB || newValue > maxB)  {  logger.LogError("Value is not in range of possible values.");  return;  }  bool add = setB.Add(newValue);  logger.Log(add.ToString());  }  else  {  logger.LogError("Wrong name of set. There are only set <a> and set <b>.");  }  }  static void ProcessCommandContains(string[] command, IsetInt a, IsetInt b, int minA, int maxA, int minB, int maxB, ILogger logger)  {  if(CkeckSetValue(command[2]) == false)  {  logger.LogError("Set can contain only integers.");  return;  }  int value = int.Parse(command[2]);  if(command[0] == "a")  {  if(value < minA || value > maxA)  {  logger.LogError("Value is not in range of possible values.");  return;  }  bool contains = a.Contains(value);  logger.Log(contains.ToString());  }  else if(command[0] == "b")  {  if(value < minB || value > maxB)  {  logger.LogError("Value is not in range of possible values.");  return;  }  bool contains = b.Contains(value);  logger.Log(contains.ToString());  }  else  {  logger.LogError("Wrong name of set. There are only set <a> and set <b>.");  }  }  static void ProcessCommandLog(string[] command, IsetInt a, IsetInt b, ILogger logger)  {  if(command[0] == "a")  {  int[] array = new int[a.GetCount()];  a.CopyTo(array);  string set = String.Join(',', array);  logger.Log(set);  }  else if(command[0] == "b")  {  int[] array = new int[b.GetCount()];  b.CopyTo(array);  string set = String.Join(',', array);  logger.Log(set);  }  else  {  logger.LogError("Wrong name of set. There are only set <a> and set <b>.");  }  }  static void ProcessCommandRead(string[] command, IsetInt a, IsetInt b, int minA, int maxA, int minB, int maxB, ILogger logger)  {  StreamReader sr = new StreamReader(command[2]);  string line = sr.ReadToEnd();  string[] array = line.Split(',');  if(command[0] != "a" && command[0] != "b")  {  logger.LogError("Wrong name of set. There are only set <a> and set <b>.");  return;  }  int counter = 0;  for(int i = 0; i < array.Length; i++)  {  if(CkeckSetValue(array[i]) == true)  {  if(command[0] == "a")  {  if(int.Parse(array[i]) < minA || int.Parse(array[i]) > maxA)  {  counter++;  }  else  {  a.Add(int.Parse(array[i]));  }  }  else if(command[0] == "b")  {  if(int.Parse(array[i]) < minB || int.Parse(array[i]) > maxB)  {  counter++;  }  else  {  b.Add(int.Parse(array[i]));  }  }  }  else  {  counter++;  }  }  if(counter != 0)  {  logger.Log("Not all values were added.");  }  else  {  logger.Log("All values were added.");  }  }  static void ProcessCommandCount(string[] command, IsetInt a, IsetInt b, ILogger logger)  {  if(command[0] == "a")  {  string count = a.GetCount().ToString();  logger.Log(count);  }  else if(command[0] == "b")  {  string count = b.GetCount().ToString();  logger.Log(count);  }  else  {  logger.LogError("Wrong name of set. There are only set <a> and set <b>.");  }  }  static void ProcessCommandRemove(string[] command, IsetInt a, IsetInt b, int minA, int maxA, int minB, int maxB, ILogger logger)  {  if(CkeckSetValue(command[2]) == false)  {  logger.LogError("Set can contain only integers.");  return;  }  int value = int.Parse(command[2]);  if(command[0] == "a")  {  if(value < minA || value > maxA)  {  logger.LogError("Value is not in range of possible values.");  return;  }  bool remove = a.Remove(value);  logger.Log(remove.ToString());  }  else if(command[0] == "b")  {  if(value < minB || value > maxB)  {  logger.LogError("Value is not in range of possible values.");  return;  }  bool remove = b.Remove(value);  logger.Log(remove.ToString());  }  else  {  logger.LogError("Wrong name of set. There are only set <a> and set <b>.");  }  }  static void ProcessCommandClear(string[] command, IsetInt a, IsetInt b, ILogger logger)  {  if(command[0] == "a")  {  int[] array = new int[a.GetCount()];  a.CopyTo(array);  for(int i = 0; i < array.Length; i++)  {  a.Remove(array[i]);  }  logger.Log("Set a is empty now");  }  else if(command[0] == "b")  {  int[] array = new int[b.GetCount()];  b.CopyTo(array);  for(int i = 0; i < array.Length; i++)  {  b.Remove(array[i]);  }  logger.Log("Set b is empty now");  }  else  {  logger.LogError("Wrong name of set. There are only set <a> and set <b>.");  }  }  static void ProcessCommandWrite(string[] command, IsetInt a, IsetInt b, ILogger logger)  {  if(command[0] == "a")  {  WriteSet(command[2], a);  logger.Log("Set a writen successfully.");  }  else if(command[0] == "b")  {  WriteSet(command[2], b);  logger.Log("Set b writen successfully.");  }  else  {  logger.LogError("Wrong name of set. There are only set <a> and set <b>.");  }  }  }  } |

|  |
| --- |
| progbase/Labs/lab1/ILogger.cs |
| namespace lab3\_2  {  public interface ILogger  {  void Log(string message);  void LogError(string errorMessage);  }  } |

|  |
| --- |
| progbase/Labs/lab1/ConsoleLogger.cs |
| using System;  namespace lab3\_2  {  public class ConsoleLogger : ILogger  {  public void Log(string message)  {  Console.WriteLine(message);  }  public void LogError(string errorMessage)  {  Console.Error.WriteLine(errorMessage);  }  }  } |

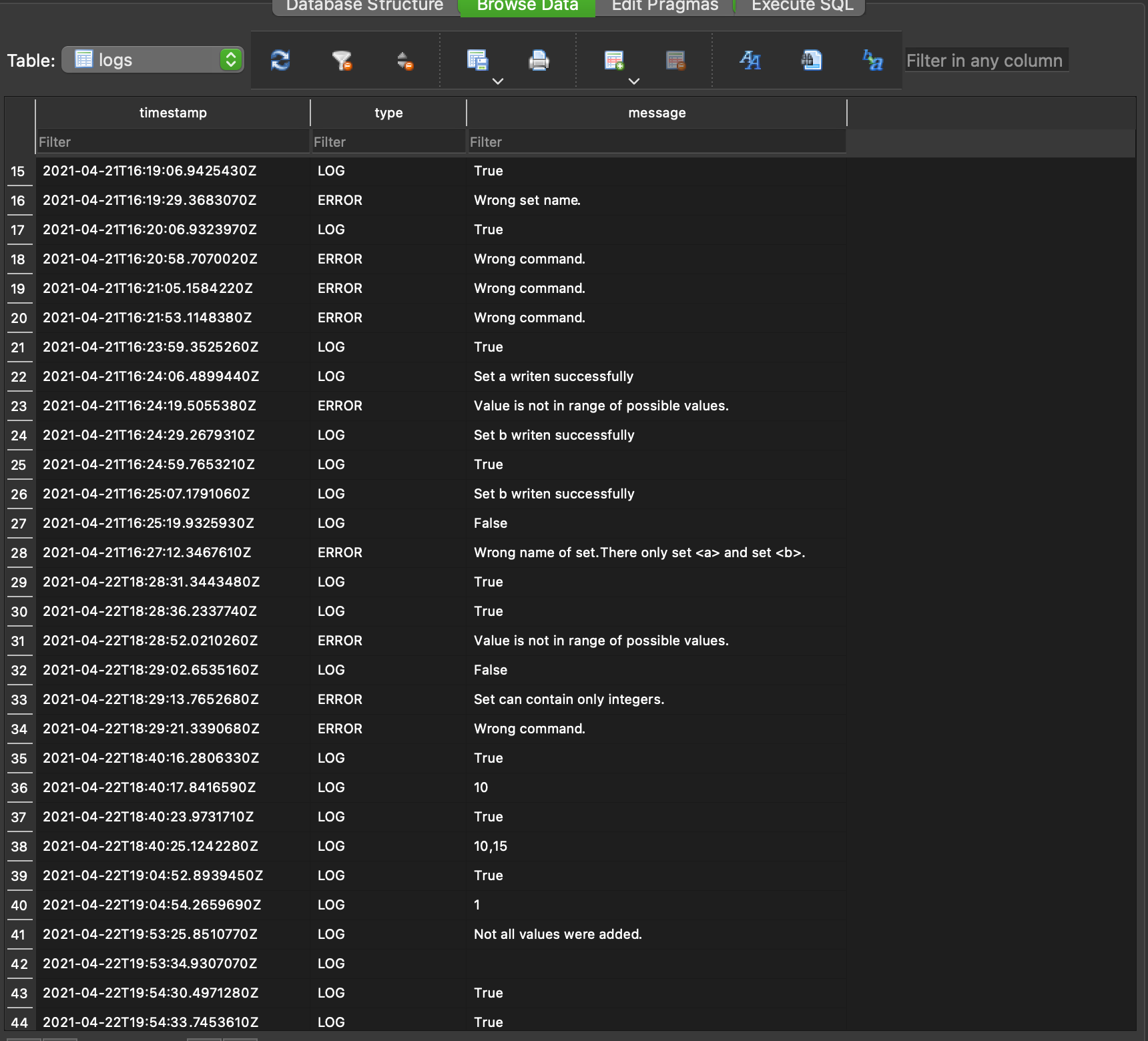
|  |
| --- |
| progbase/Labs/lab1/SQLLogger.cs |
| using System;  using Microsoft.Data.Sqlite;  namespace lab3\_2  {  public class SQLLogger : ILogger  {  private string \_filePath;  public SQLLogger(string path)  {  this.\_filePath = path;  }  public void Log(string message)  {  SqliteConnection connection = new SqliteConnection($"Data Source = {\_filePath}");  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText =  @"  INSERT INTO logs (timestamp, type, message)  VALUES ($timestamp, $type, $message);  SELECT last\_insert\_rowid();  ";  string type = "LOG";  command.Parameters.AddWithValue("$timestamp", DateTime.UtcNow.ToString("o"));  command.Parameters.AddWithValue("$type", type);  command.Parameters.AddWithValue("$message", message);  long newId = (long)command.ExecuteScalar();  connection.Close();  }  public void LogError(string errorMessage)  {  SqliteConnection connection = new SqliteConnection($"Data Source = {\_filePath}");  connection.Open();  SqliteCommand command = connection.CreateCommand();  command.CommandText =  @"  INSERT INTO logs (timestamp, type, message)  VALUES ($timestamp, $type, $message);  SELECT last\_insert\_rowid();  ";  string type = "ERROR";  command.Parameters.AddWithValue("$timestamp", DateTime.UtcNow.ToString("o"));  command.Parameters.AddWithValue("$type", type);  command.Parameters.AddWithValue("$message", errorMessage);  long newId = (long)command.ExecuteScalar();  connection.Close();  }  }  } |

|  |
| --- |
| progbase/Labs/lab1/ArraySetInt.cs |
| using System;  using static System.Math;  namespace lab3\_2  {  public class ArraySetInt : IsetInt  {  /\*  Set based on array of boolean values, which makes some methods faster, but it can contain only limited number of integers  User can set range of possible values  in case of union of two sets, lowest and largest borders of two sets will set new range of values  \*/  private bool[] \_items;  private int \_size;  private int \_minValue;  private int \_maxValue;  public ArraySetInt(int minValue, int maxValue)  {  if(minValue < 0 && maxValue < 0)  {  this.\_items = new bool[Math.Abs(minValue) - Math.Abs(maxValue) + 1];  }  else if(minValue < 0 && maxValue >= 0)  {  this.\_items = new bool[Math.Abs(minValue) + maxValue + 1];  }  else if(minValue >= 0 && maxValue >= 0)  {  this.\_items = new bool[maxValue - minValue + 1];  }  this.\_size = 0;  this.\_minValue = minValue;  this.\_maxValue = maxValue;  }  public bool Add(int value)  {  bool isHere = this.Contains(value);  if(isHere)  {  return false;  }  if(this.\_minValue >= 0)  {  this.\_items[value - this.\_minValue] = true;  }  else  {  this.\_items[value + Math.Abs(this.\_minValue)] = true;  }  \_size++;  return true;  }  public void Clear()  {  for(int i = 0; i < this.\_items.Length; i++)  {  this.\_items[i] = false;  }  this.\_size = 0;  }  public bool Contains(int value)  {  if(this.\_minValue >= 0)  {  if(this.\_items[value - this.\_minValue] == true)  {  return true;  }  else  {  return false;  }  }  else  {  if(this.\_items[value + Math.Abs(this.\_minValue)] == true)  {  return true;  }  else  {  return false;  }  }  }  public void CopyTo(int[] array)  {  int counter = 0;  if(this.\_minValue >= 0)  {  for(int i = 0; i < this.\_items.Length; i++)  {  if(this.\_items[i] == true)  {  array[counter] = i + this.\_minValue;  counter++;  }  }  }  else  {  for(int i = 0; i < this.\_items.Length; i++)  {  if(this.\_items[i] == true)  {  array[counter] = i - Math.Abs(this.\_minValue);  counter++;  }  }  }  }  public int GetCount()  {  return \_size;  }  public bool Remove(int value)  {  if(this.\_minValue >= 0)  {  if(this.\_items[value - this.\_minValue] == false)  {  return false;  }  else  {  this.\_items[value - this.\_minValue] = false;  this.\_size--;  return true;  }  }  else  {  if(this.\_items[value + Math.Abs(this.\_minValue)] == false)  {  return false;  }  else  {  this.\_items[value + Math.Abs(this.\_minValue)] = false;  this.\_size--;  return true;  }  }  }  public bool SetEquals(IsetInt other)  {  int[] array1 = new int[this.GetCount()];  this.CopyTo(array1);  int[] array2 = new int[other.GetCount()];  other.CopyTo(array2);  if(array1.Length != array2.Length)  {  return false;  }  for(int i = 0; i < array1.Length; i++)  {  if(array1[i] != array2[i])  {  return false;  }  }  return true;  }  public void UnionWith(IsetInt other)  {  int[] array2 = new int[other.GetCount()];  other.CopyTo(array2);  int min2 = array2[0];  int max2 = array2[0];  for(int i = 0; i < array2.Length; i++)  {  if(array2[i] < min2)  {  min2 = array2[i];  }  if(array2[i] > max2)  {  max2 = array2[i];  }  }  if(min2 < this.\_minValue && max2 > this.\_maxValue)  {  if(min2 < 0 && max2 < 0)  {  Array.Resize(ref this.\_items, Math.Abs(min2) - Math.Abs(max2) + 1);  this.\_minValue = min2;  this.\_maxValue = max2;  }  if(min2 < 0 && max2 >= 0)  {  Array.Resize(ref this.\_items, Math.Abs(min2) + max2 + 1);  this.\_minValue = min2;  this.\_maxValue = max2;  }  if(min2 >= 0 && max2 >= 0)  {  Array.Resize(ref this.\_items, max2 - min2 + 1);  this.\_minValue = min2;  this.\_maxValue = max2;  }  }  else if(min2 < this.\_minValue && max2 <= this.\_maxValue)  {  if(min2 < 0 && this.\_maxValue < 0)  {  Array.Resize(ref this.\_items, Math.Abs(min2) - Math.Abs(this.\_maxValue) + 1);  this.\_minValue = min2;  }  if(min2 < 0 && this.\_maxValue >= 0)  {  Array.Resize(ref this.\_items, Math.Abs(min2) + this.\_maxValue + 1);  this.\_minValue = min2;  }  if(min2 >= 0 && this.\_maxValue >= 0)  {  Array.Resize(ref this.\_items, this.\_maxValue - min2 + 1);  this.\_minValue = min2;  }  }  else if(min2 >= this.\_minValue && max2 > this.\_maxValue)  {  if(this.\_minValue < 0 && max2 < 0)  {  Array.Resize(ref this.\_items, Math.Abs(this.\_minValue) - Math.Abs(max2) + 1);  this.\_maxValue = max2;  }  if(this.\_minValue < 0 && max2 >= 0)  {  Array.Resize(ref this.\_items, Math.Abs(this.\_minValue) + max2 + 1);  this.\_maxValue = max2;  }  if(this.\_minValue >= 0 && max2 >= 0)  {  Array.Resize(ref this.\_items, max2 - this.\_minValue+ 1);  this.\_maxValue = max2;  }  }  for(int i = 0; i < array2.Length; i++)  {  this.Add(array2[i]);  }  }  }  } |

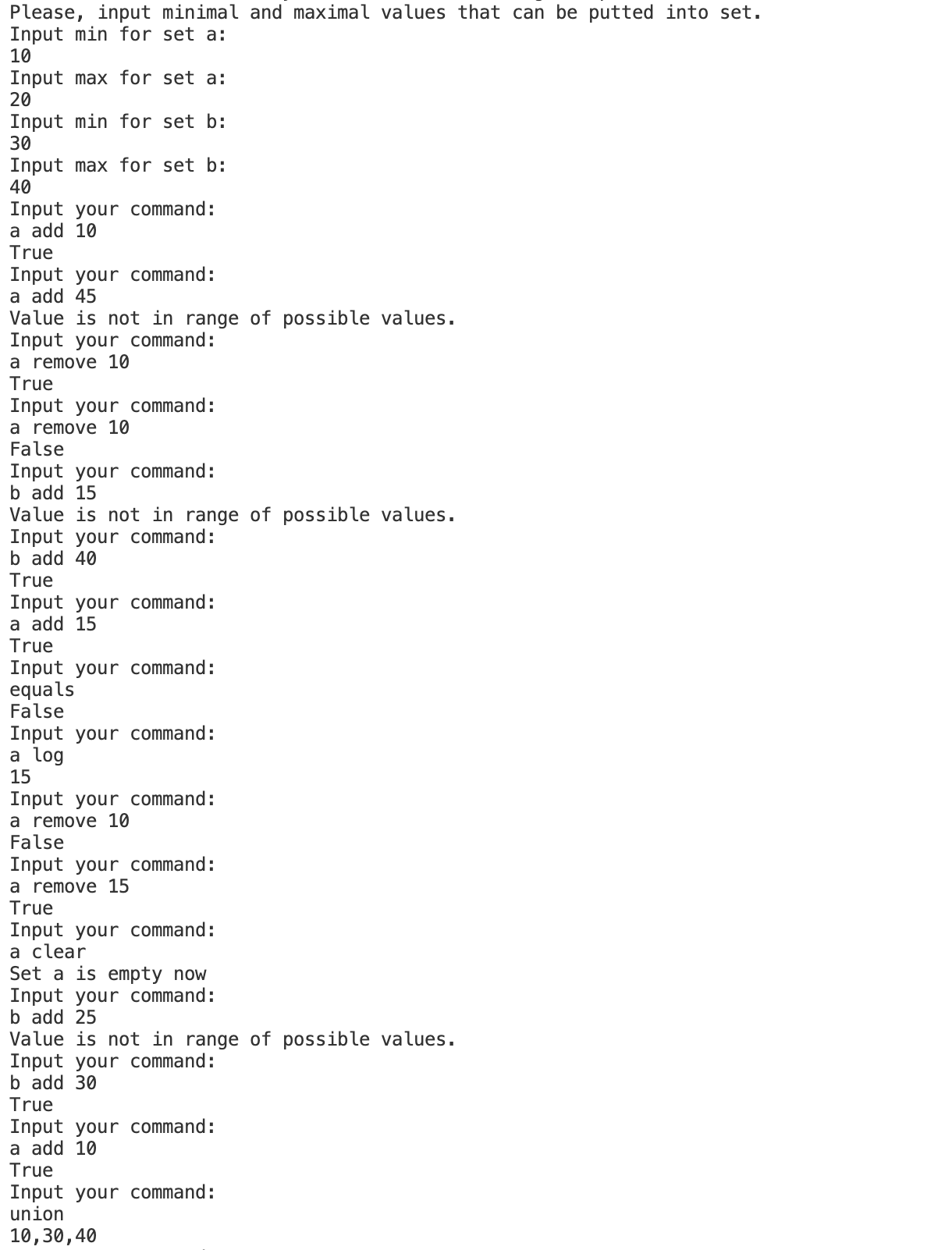
|  |
| --- |
| progbase/Labs/lab1/ISetInt.cs |
| namespace lab3\_2  {  public interface IsetInt  {  int GetCount();  bool Add(int value);  bool Remove(int value);  bool Contains(int value);  void Clear();  void CopyTo(int[] array);  bool SetEquals(IsetInt other);  void UnionWith(IsetInt other);  }  } |

**Результати**

Приклад результату роботи програми при логуванні у SQL таблицю:



Приклад результату роботи програми при логуванні у консоль:



**Висновки**

**Виконавши дану лабораторну роботу код програми був розділений на модулі. Використані інтерфейси для розділення коду клієнта та реалізації та забезпечення змінності реалізації.**