Лабораторна робота №3

Тема

Модульне програмування і контракти

Мета

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

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 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 і B одну із операцій за варіантом (див. **додаток A**) 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 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.

Методичні вказівки

Рекомендовані 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):

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

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
```

```
{
   bool 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 А**).

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 В**.

Контрольні питання

1. Що таке модуль? В чому полягає модульне програмування?
2. Опишіть АТД Множина. Які операції підтримує дана АТД? Назвіть способи реалізації АТД Множина.
3. Що таке контракт? Як реалізувати контракт? Як і для чого можна використати контракт у коді?

Додатки

Додаток А. Варіант додаткових методів множини

n - номер у списку групи.

Таблиця 1. Варіант першого метода

n % 4	**Методи (див. опис у таблиці 3)**
0	`SetEquals`
1	`IsSubsetOf`
2	`IsSupersetOf`
3	`Overlaps`

n - номер у списку групи.

Таблиця 2. Варіант другого метода

n % 3	**Методи (див. опис у таблиці 3)**
0	`SymmetricExceptWith`
1	`UnionWith`

2	`IntersectWith`

Поточна множина - **this**, інша множина - **other**:

Таблиця 3. Опис додаткових методів множини

Метод	**Опис**
`bool SetEquals(ISetInt other)`	Перевіряє чи поточна та інша множини містять однаковий набір елементів.
`bool IsSubsetOf(ISetInt other)`	Перевіряє чи поточна множина є підмножиною іншої множини
`bool IsSupersetOf(ISetInt other)`	Перевіряє чи поточна множина є надмножиною іншої множини
`bool Overlaps(ISetInt other)`	Перевіряє чи поточна множина та інша множина мають спільні елементи
`void SymmetricExceptWith(ISetInt other)`	Модифікує поточну множину так, що вона містить тільки елементи, що є або у цій або у іншій множині, але не є спільними для цієї і іншої множини одночасно
`void UnionWith(ISetInt other)`	Модифікує поточну множину так, що вона містить всі елементи з цієї та іншої множини.
`void IntersectWith(ISetInt other)`	Модифікує поточну множину так, що вона містить тільки елементи, що є спільними для цієї і іншої множини

Додаток B. Варіант способу запису чисел у файлах

n - номер у списку групи.

Таблиця 4. Варіант способу запису чисел у файлах

n % 3	**Спосіб**
0	Кожне число з нового рядка
1	Числа у файлі записані через коми
2	Числа у файлі записані через пробіли

Додаток C. Варіант класу логування

n - номер у списку групи.

Таблиця 5. Варіант класу логування

n % 5	Клас (див. опис у таблиці 6)
0	`PlainFileLogger`
1	`ChunkPlainFileLogger`
2	`CsvFileLogger`
3	`CsvFileLogger2`
4	`SqliteFileLogger`

Таблиця 6. Опис класів логування

Клас	Опис
`ConsoleLogger`	Писати повідомлення через `Console.WriteLine("Message")`, писати помилки через `Console.Error.WriteLine("Error message")`
`PlainFileLogger`	Писати повідомлення і помилки у два текстові файли. Писати повідомлення у перший файл, писати помилки у другий файл. Повідомлення у файлах розділяти одним пустим рядком. Шляхи до файлів задавати через конструктор.
`ChunkPlainFileLogger`	Писати повідомлення і помилки у текстовий файл. Якщо кількість рядків, що записані у текстовий файл перевищує певне порогове значення (задається через конструктор), створити новий файл і писати нові записи у нього. Називати файли за датою їх створення, наприклад: `DateTime.Now.ToString("yyyy-MM-ddTHH:mm:ss.fff", System.Globalization.CultureInfo.InvariantCulture) + ".txt"` `->` **`2021-03-01T09:42:36.153.txt`** Шлях до директорії з файлами і максимальну кількість рядків у кожному файлі передавати через конструктор.
`CsvFileLogger`	Писати повідомлення і помилки у один файл у форматі CSV. CSV таблиця має містити три стовпці: **`timestamp,type,message`**. У першому стовпці записувати рядок часу у форматі ISO 8601, у другому тип повідомлення (`LOG`|`ERROR`), у третьому саме повідомлення. Якщо CSV неекрановане, рекомендується перед записом повідомлення видаляти символи, що потребують екранування. Шлях до файлу задавати через конструктор.
`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.

`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, у другому тип повідомлення (`LOG`|`ERROR`), у третьому саме повідомлення. Шлях до файлу задавати через конструктор.