Лабораторна робота №2

Тема: Використання методів розширень та узагальнень у C#.

Мета роботи: навчитися використовувати методи розширення та узагальнення у мові програмування C#.

Виділений час: 12 годин (4 години лабораторних робіт та 8 годин самостійної роботи).

Завдання:

1. **Опрацювати теорію**:
    - Відеозапис лекцій №03-04:
      Лекція 03-04. Статичний імпорт класів, локальні функції, узагальнення, Nullable-типи.

   Довідкова інформація:

    - Документація: make-class-foreach-statement
    - Документація: using-indexers
2. **Реалізувати методи розширення**:
    - **Для класу String**:
        - Інвертування рядка.
        - Підрахунок кількості входжень заданого у параметрі символа у рядок.
    - **Для одновимірних масивів**:
        - Метод, що визначає, скільки разів зустрічається задане значення у масиві (метод має працювати для одновимірних масивів усіх типів, для реалізації використати узагальнення та їх обмеження за допомогою where).
        - Метод, що повертає новий масив такого ж типу і формує його з унікальних елементів (видаляє повтори).
    - **Написати код для демонстрації роботи реалізованих методів розширення.**

3. **Реалізувати узагальнені класи**:
    - Реалізувати узагальнений клас для зберігання “розширеного словника” (для ключа передбачається два значення).

      csharp
      Copy code
      ExtendedDictionary<T, U, V>

      де T - тип даних ключа, U - тип даних першого значення, V - тип даних другого значення.

      Передбачити операції:

      - Додавання елемента у словник.
      - Видалення елемента з словника за заданим ключем.
      - Перевірка наявності елемента із заданим ключем.
      - Перевірка наявності елемента із заданим значенням (значення1 та значення2).
      - Повернення елемента за заданим ключем (реалізувати операцію індексування).
      - Властивість, що повертає кількість елементів.

      Представлення елемента словника:
      Реалізувати у вигляді окремого класу ExtendedDictionaryElement<T, U, V>, передбачивши властивості для доступу до ключа, першого та другого значення.

      Вимоги:
      Словник повинен мати можливість використання у циклах foreach:

      csharp
      Copy code
      foreach (var elem in array) { ... }

- **Написати код для демонстрації роботи з реалізованими узагальненими класами.**

4. **Запушити виконану роботу у репозиторій на GitHub**:
   - Репозиторій назвати DotNetLab2.
   - Відкрити доступ для викладачів:
     - morozov@ztu.edu.ua
     - 4ov.ztu@gmail.com
5. **Оцінювання**:
   - Для отримання оцінки за даною лабораторною роботою проходження код-рев'ю не передбачається (воно буде у наступній :)).

Код додатку:

```
using System;
using System.Collections.Generic;
using System.Linq;

// Extension Methods
public static class Extensions
{
    // Reverse a string
    public static string ReverseString(this string str)
    {
        return new string(str.Reverse().ToArray());
    }

    // Count occurrences of a character in a string
    public static int CountOccurrences(this string str, char c)
    {
        return str.Count(ch => ch == c);
    }

    // Count occurrences of an element in an array
    public static int CountOccurrences<T>(this T[] array, T value) where T : IEquatable<T>
    {
        return array.Count(item => item.Equals(value));
    }

    // Create a unique array from the original array
    public static T[] GetUniqueElements<T>(this T[] array)
    {
        return array.Distinct().ToArray();
    }
}

// Generic ExtendedDictionary
public class ExtendedDictionaryElement<T, U, V>
{
    public T Key { get; set; }
    public U Value1 { get; set; }
    public V Value2 { get; set; }

    public ExtendedDictionaryElement(T key, U value1, V value2)
```

```
    {
        Key = key;
        Value1 = value1;
        Value2 = value2;
    }
}

public class ExtendedDictionary<T, U, V> : IEnumerable<ExtendedDictionaryElement<T, U, V>>
{
    private readonly List<ExtendedDictionaryElement<T, U, V>> _elements = new();

    public void Add(T key, U value1, V value2)
    {
        _elements.Add(new ExtendedDictionaryElement<T, U, V>(key, value1, value2));
    }

    public bool Remove(T key)
    {
        var element = _elements.FirstOrDefault(e => EqualityComparer<T>.Default.Equals(e.Key, key));
        if (element != null)
        {
            _elements.Remove(element);
            return true;
        }
        return false;
    }

    public bool ContainsKey(T key)
    {
        return _elements.Any(e => EqualityComparer<T>.Default.Equals(e.Key, key));
    }

    public bool ContainsValue(U value1, V value2)
    {
        return _elements.Any(e => EqualityComparer<U>.Default.Equals(e.Value1, value1) &&
EqualityComparer<V>.Default.Equals(e.Value2, value2));
    }

    public ExtendedDictionaryElement<T, U, V> this[T key]
    {
        get => _elements.First(e => EqualityComparer<T>.Default.Equals(e.Key, key));
    }

    public int Count => _elements.Count;

    public IEnumerator<ExtendedDictionaryElement<T, U, V>> GetEnumerator()
    {
        return _elements.GetEnumerator();
    }

    System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}

// Demonstration
class Program
{
    static void Main()
    {
        // String extension methods
        string example = "hello world";
        Console.WriteLine(example.ReverseString());
        Console.WriteLine(example.CountOccurrences('o'));

        // Array extension methods
        int[] numbers = { 1, 2, 2, 3, 3, 3 };
        Console.WriteLine(numbers.CountOccurrences(3));
```

```
        Console.WriteLine(string.Join(", ", numbers.GetUniqueElements()));

        // Generic dictionary demonstration
        var dictionary = new ExtendedDictionary<int, string, string>();
        dictionary.Add(1, "Value1", "Extra1");
        dictionary.Add(2, "Value2", "Extra2");
        Console.WriteLine(dictionary.ContainsKey(1));
        Console.WriteLine(dictionary.ContainsValue("Value1", "Extra1"));

        foreach (var element in dictionary)
        {
            Console.WriteLine($"Key: {element.Key}, Value1: {element.Value1}, Value2: {element.Value2}");
        }
    }
}
```