**15 Обобщения**

Задание 1. Создайте класс MyList<T>. Реализуйте возможность использования его экземпляра аналогично экземпляру класса List<T>. Минимально требуемый интерфейс взаимодействия с экземпляром, должен включать метод добавления элемента, индексатор для получения значения элемента по указанному индексу и свойство только для чтения для получения общего количества элементов. Создайте расширяющий метод: public static T[] GetArray<T>(this MyList<T> list). Примените расширяющий метод к экземпляру типа MyList<T>. Выведите на экран значения элементов массива, который вернул расширяющий метод GetArray().

Листинг программы:

namespace Space

{

// Объявляем обобщенный класс MyList, параметризированный типом T

public class MyList<T>

{

private T[] \_items;

private int \_count;

// Конструктор класса

public MyList()

{

const int defaultCapacity = 4;

\_items = new T[defaultCapacity];

}

// Метод для добавления элемента в конец списка

public void Add(T item)

{

// Если массив заполнен, увеличиваем его размер

if (\_count == \_items.Length)

{

EnsureCapacity(\_count + 1);

}

// Добавляем элемент в конец списка и увеличиваем количество элементов

\_items[\_count++] = item;

}

// Индексатор для получения и установки значения элемента по индексу

public T this[int index]

{

get

{

// Проверяем, что индекс находится в допустимых границах массива

if (index < 0 || index >= \_count)

{

// выбрасываем исключение, если индекс некорректен

throw new ArgumentOutOfRangeException(nameof(index));

}

// возвращаем элемент с указанным индексом

return \_items[index];

}

set

{

if (index < 0 || index >= \_count)

{

throw new ArgumentOutOfRangeException(nameof(index));

}

// устанавливаем элемент с указанным индексом

\_items[index] = value;

}

}

// Свойство только для чтения, возвращающее количество элементов в списке

public int Count

{

get { return \_count; }

}

// Приватный метод для увеличения ёмкости массива элементов

private void EnsureCapacity(int minCapacity)

{

// Рассчитываем новую ёмкость массива

int newCapacity = \_items.Length == 0 ? 4 : \_items.Length \* 2;

if (newCapacity < minCapacity)

{

newCapacity = minCapacity;

}

// Изменяем размер массива элементов

Array.Resize(ref \_items, newCapacity);

}

}

// создаём расширяющий метод

static class MyListExtensions

{

public static T[] GetArray<T>(this MyList<T> list)

{

T[] result = new T[list.Count];

for (int i = 0; i < list.Count; i++)

{

result[i] = list[i];

}

return result;

}

}

class Program

{

static void Main()

{

var myList = new MyList<int>();

myList.Add(1);

myList.Add(2);

myList.Add(3);

Console.WriteLine($"Количество элементов в списке: {myList.Count}");

Console.WriteLine($"Первый элемент в списке: {myList[0]}");

myList[2] = 4;

Console.WriteLine($"Третий элемент в списке: {myList[2]}");

int[] array = myList.GetArray();

Console.Write("Элементы массива: ");

foreach (int item in array)

{

Console.Write(item + " ");

}

}

}

}

Анализ результатов:

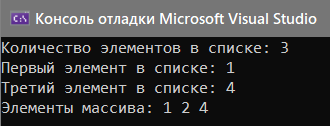


Рисунок 15.1 – Результат работы программы

Задание 2. Создайте класс MyDictionary <TKey, TValue>. Реализуйте возможность использования его экземпляра аналогично экземпляру класса Dictionary.

Минимально требуемый интерфейс взаимодействия с экземпляром, должен включать метод добавления пар элементов, индексатор для получения значения элемента по указанному индексу и свойство только для чтения для получения общего количества пар элементов.

Листинг программы:

namespace Space

{

// Обобщенный класс MyDictionary<TKey, TValue> представляет собой реализацию словаря, похожую на класс Dictionary<TKey, TValue>.

public class MyDictionary<TKey, TValue>

{

// Класс MyDictionary использует словарь Dictionary<TKey, TValue> для хранения элементов.

private readonly Dictionary<TKey, TValue> dictionary = new Dictionary<TKey, TValue>();

// Метод Add добавляет элемент с указанным ключом и значением

// в словарь.

public void Add(TKey key, TValue value)

{

dictionary.Add(key, value);

}

// Индексатор this позволяет получить значение элемента словаря по указанному ключу или задать новое значение для существующего элемента.

public TValue this[TKey key]

{

get { return dictionary[key]; }

set { dictionary[key] = value; }

}

// Свойство Count позволяет получить общее количество элементов в словаре.

public int Count

{

get { return dictionary.Count; }

}

// Метод для вывода всех элементов словаря.

public void PrintAllElements()

{

foreach (KeyValuePair<TKey, TValue> pair in dictionary)

{

Console.WriteLine("{0} : {1}", pair.Key, pair.Value);

}

}

}

class Program

{

static void Main(string[] args)

{

// Создаем новый экземпляр класса MyDictionary<string, int>.

var myDictionary = new MyDictionary<string, int>();

// Добавляем несколько элементов в словарь.

myDictionary.Add("один", 1);

myDictionary.Add("два", 2);

myDictionary.Add("три", 3);

// Выводим все элементы словаря.

myDictionary.PrintAllElements();

Console.WriteLine(myDictionary["два"]);

myDictionary["три"] = 33;

Console.WriteLine(myDictionary["три"]);

// Выводим общее количество элементов в словаре.

Console.WriteLine(myDictionary.Count);

}

}

}

Анализ результатов:

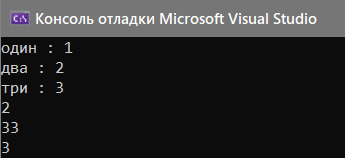


Рисунок 15.2 – Результат работы программы