**Занятие № 14**

**Дата выполнения работы:** 05.05.2023

# **Тема работы:** «Обобщенные классы»

**Ход работы**

**Задание 1**

Вариант 1

Описать класс для работы с одномерным массивом:

- конструктор, выделяющий память под заданное в его параметре количество

элементов массива;

- конструктор, заполняющий заданное в его параметре количество элементов

массива значениями членов ряда Тейлора для функции ln x для заданного х.

- свойство, доступное только для чтения, для получения количества элементов

массива, меньших 0,2;

- метод, вычисляющий сумму модулей элементов, расположенных до (левее)

первого элемента, равного нулю

Вывод на экран выполнять только в методе Main класса-клиента. Программа

должна адекватно реагировать на ошибки пользователя и различные варианты

исходных данных. Все тестовые данные предъявить преподавателю.

**Листинг программы:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace App1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

try

{

// Создаем массив из 5 элементов и выводим количество элементов меньше 0.2

ArrayHandler arr1 = new ArrayHandler(5);

textBox1.Text = ($"Количество элементов меньше 0.2: {arr1.CountElementsLessThan0\_2}");

// Создаем массив из 7 элементов, заполненный значениями членов ряда Тейлора для ln(x) при x=3

ArrayHandler arr2 = new ArrayHandler(7, 3);

textBox2.Text = ($"Количество элементов меньше 0.2: {arr2.CountElementsLessThan0\_2}");

// Выводим сумму модулей элементов, расположенных до первого нуля

textBox3.Text = ($"Сумма модулей элементов, расположенных до первого нуля: {arr2.SumOfModulusBeforeFirstZero()}");

}

catch (ArgumentException ex)

{

textBox1.Text = ($"Ошибка: {ex.Message}");

}

catch (Exception ex)

{

textBox1.Text = ($"Произошла ошибка: {ex.Message}");

}

}

}

public class ArrayHandler

{

private double[] array;

// Конструктор, выделяющий память под заданное количество элементов массива

public ArrayHandler(int length)

{

if (length <= 0)

throw new ArgumentException("Длина массива должна быть положительным числом");

array = new double[length];

}

// Конструктор, заполняющий заданное количество элементов массива значениями членов ряда Тейлора для функции ln x для заданного x

public ArrayHandler(int length, double x)

{

if (length <= 0)

throw new ArgumentException("Длина массива должна быть положительным числом");

if (x <= 0)

throw new ArgumentException("Аргумент функции ln(x) должен быть положительным числом");

array = new double[length];

for (int i = 0; i < length; i++)

{

array[i] = Math.Pow(-1, i) \* Math.Pow(x - 1, i + 1) / (i + 1);

}

}

// Свойство, доступное только для чтения, для получения количества элементов массива, меньших 0.2

public int CountElementsLessThan0\_2

{

get

{

int count = 0;

for (int i = 0; i < array.Length; i++)

{

if (array[i] < 0.2)

{

count++;

}

}

return count;

}

}

// Метод, вычисляющий сумму модулей элементов, расположенных до (левее) первого элемента, равного нулю

public double SumOfModulusBeforeFirstZero()

{

double sum = 0;

for (int i = 0; i < array.Length; i++)

{

if (array[i] == 0)

{

break;

}

sum += Math.Abs(array[i]);

}

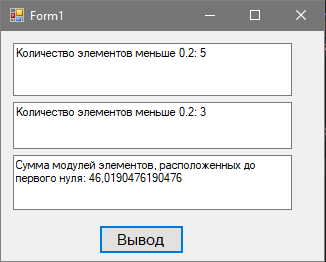
return sum;

}

}

}

**Результат:**

****

**Задание 2**

Вариант 1

Создать обобщенную коллекцию Stack<T> и заполнить ее данными, тип которых char. Вывести коллекцию на консоль Удалите из коллекции n последовательных элементов Добавьте другие элементы (используйте все возможные методы добавления для вашего типа коллекции). Создайте вторую коллекцию List<T> и заполните ее данными из первой коллекции. Выведите вторую коллекцию на консоль. В случае не совпадения количества параметров (например, LinkedList<T> и Dictionary<Tkey, TValue>), при нехватке - генерируйте ключи, в случае избыточности – оставляйте TValue. Найдите во второй коллекции заданное значение.

**Листинг программы:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace App2

{

public partial class Form1 : Form

{

private Stack<char> charStack = new Stack<char>();

private List<char> charList = new List<char>();

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

charStack.Push('a');

charStack.Push('b');

charStack.Push('c');

charStack.Push('d');

charStack.Push('e');

DisplayStack();

}

private void DisplayStack()

{

char[] charArray = charStack.ToArray();

string stackString = new string(charArray);

StackLabel.Text = stackString;

}

private void button2\_Click(object sender, EventArgs e)

{

int n = 2;

for (int i = 0; i < n; i++)

{

charStack.Pop();

}

DisplayStack();

}

private void button3\_Click(object sender, EventArgs e)

{

charStack.Push('f');

charStack.Push('g');

charStack.Push('h');

DisplayStack();

}

private void button4\_Click(object sender, EventArgs e)

{

charList.Add('f');

charList.Add('g');

charList.Add('h');

DisplayList();

}

private void DisplayList()

{

string listString = new string(charList.ToArray());

ListLabel.Text = listString;

}

private void button5\_Click(object sender, EventArgs e)

{

int n = 2;

charList.RemoveRange(charList.Count - n, n);

DisplayList();

}

private void button6\_Click(object sender, EventArgs e)

{

charList.Add('a');

charList.Add('b');

charList.Add('c');

charList.Add('d');

charList.Add('e');

DisplayList();

}

private void button7\_Click(object sender, EventArgs e)

{

charList.Clear();

charList.AddRange(charStack);

DisplayList();

}

private void button8\_Click(object sender, EventArgs e)

{

char searchChar = 'c';

int index = charList.IndexOf(searchChar);

if (index == -1)

{

MessageBox.Show($"Character '{searchChar}' not found in the list.");

}

else

{

MessageBox.Show($"Character '{searchChar}' found at index {index} in the list.");

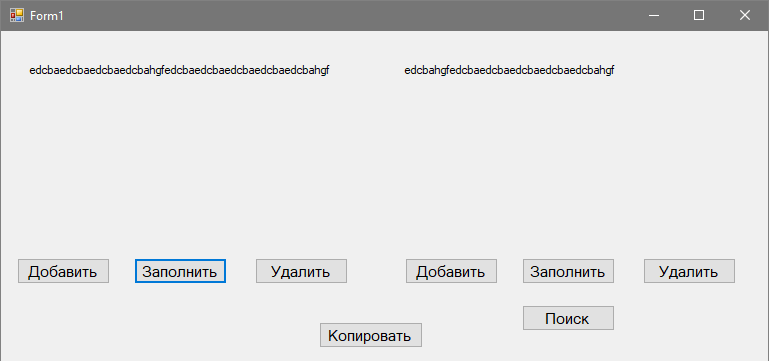
}

}

}

}

**Результат:**

****

**Вывод:** В C# обобщенные коллекции позволяют хранить и обрабатывать элементы различных типов, без необходимости явного приведения типов и повышения безопасности во время компиляции. Обобщенные коллекции предоставляют широкий спектр структур данных, таких как списки, стеки, очереди, словари и т. д. Каждый тип коллекции предоставляет определенный функционал, позволяющий удобно и эффективно работать с данными. Благодаря использованию обобщений, можно создавать типобезопасные коллекции, которые улучшают безопасность и производительность приложения, а также упрощают процесс разработки. Кроме того, обобщенные коллекции являются одним из основных инструментов программиста в C#, поскольку они обеспечивают универсальность и переносимость кода.