**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА (САМАРСКИЙ УНИВЕРСИТЕТ)»**

Институт Информатики и кибернетики
Кафедра Программных систем

# ОТЧЁТ

## по лабораторной работе

№4 «Язык программирования C#: Стандартные интерфейсы»

по дисциплине «Языки программирования и структуры данных»

Выполнил Фадеев А.М.

Проверил Котенёва С.Э.

Самара

2024

# ЗАДАНИЕ

Задание 0.

Прочитать теоретический материал.

Задание 1.

В классах ArrayVector и LinkListVector переопределить унаследованный от класса Object метод Equals() таким образом, чтобы он сравнивал на равенство любой объект типа IVectorable. Вектора считаются равными, если они равны и по числу координат, и покоординатно. Также можно переопределить метод GetHashCode().

Задание 2.

Сделать классы ArrayVector и LinkListVector реализующими интерфейс IComparable, и реализовать в них метод CompareTo() – метод сравнивает вектора типа IVectorable по числу их координат.

Описать дополнительный класс, реализующий интерфейс IComparer. Реализовать метод Compare() интерфейса IComparer, который сравнивает два вектора типа IVectorable по их модулю.

В методе Main() класса Program создать массив векторов (ссылок типа интерфейс), хранящий вперемежку вектора разного типа. Найти в этом массиве вектора с минимальным и максимальным числом координат, используя метод CompareTo().

Отсортировать массив векторов по возрастанию их модулей, используя метод Compare().

Задание 3.

Сделать классы ArrayVector и LinkListVector реализующими интерфейс ICloneable и реализовать в них метод Clone(), выполняющий глубокое клонирование объектов.

В методе Main() класса Program выбрать один из векторов в массиве, выполнить его клонирование, продемонстрировать результат клонирования (например, изменив один из векторов – клонируемый или клон – и вывести

на экран оба вектора для сравнения, возможно также использование метода Equals()).

Задание 4.

Протестировать работу приложения в классе Program, разработать адекватный интерфейс пользователя. Необходимо отлавливать и обрабатывать все возможные исключения.

Воспользоваться пользовательским интерфейсом из лабораторной работы 3.

Задание 5.

Подготовить отчет о работе.

# КОД ПРОГРАММЫ

```
namespace Lab04;

public static class Program
{
    public static void Main()
    {
        List<IVectorable> vectors = new List<IVectorable>();
        string inp;

        LogVectors(vectors);

        while (true)
        {
            Console.WriteLine("Выберете действие:\n\n" +
                              "\t1  - Сумма векторов\n" +
                              "\t2  - Скалярное умножение\n" +
                              "\t3  - Умножение на число\n" +
                              "\t4  - Рассчитать модуль вектора\n" +
                              "\t5  - Добавить вектор в список\n" +
                              "\t6  - Удалить вектор из списка\n" +
                              "\t7  - Клонировать вектор\n" +
                              "\t8    - Вывести вектора с минимальным и
максимальным количеством координат\n" +
                              "\t9    - Отсортировать вектора по количеству
координат\n" +
                              "\t10 - Отсортировать вектора по модулю\n" +
                              "\t0  - Выход\n");

            inp = Console.ReadLine();

            switch (inp)
            {
                case "0":
                    Console.WriteLine("Выход из программы...");
                    return;
                case "1":
                {
                    LogVectors(vectors);

                    int firstVectorIdx, secondVectorIdx;
```

```
                    do
                    {
                        Console.Write("Введите индекс первого вектора: ");
                        inp = Console.ReadLine();
                    } while (!int.TryParse(inp, out firstVectorIdx) ||
firstVectorIdx <= 0 || firstVectorIdx > vectors.Count);

                    do
                    {
                        Console.Write("Введите индекс второго вектора: ");
                        inp = Console.ReadLine();
                    } while (!int.TryParse(inp, out secondVectorIdx) ||
secondVectorIdx <= 0 || secondVectorIdx > vectors.Count);

                    try
                    {
                        var result = Vectors.Sum(vectors[firstVectorIdx - 1],
vectors[secondVectorIdx - 1]);
                        result.Log($"Результат сложения {firstVectorIdx}-го и
{secondVectorIdx}-го векторов");
                    }
                    catch (Exception e)
                    {
                        Console.WriteLine("Длины векторов не совпадают");
                    }

                    break;
                }
                case "2":
                {
                    LogVectors(vectors);

                    int firstVectorIdx, secondVectorIdx;

                    do
                    {
                        Console.Write("Введите индекс первого вектора: ");
                        inp = Console.ReadLine();
                    } while (!int.TryParse(inp, out firstVectorIdx) ||
firstVectorIdx <= 0 || firstVectorIdx > vectors.Count);

                    do
                    {
                        Console.Write("Введите индекс второго вектора: ");
```

```
                    inp = Console.ReadLine();
                } while (!int.TryParse(inp, out secondVectorIdx) ||
secondVectorIdx <= 0 || secondVectorIdx > vectors.Count);

                try
                {
                    var result =
Vectors.ScalarMultiply(vectors[firstVectorIdx - 1], vectors[secondVectorIdx -
1]);
                    Console.WriteLine($"Результат скалярного умножения:
{result}");
                }
                catch (Exception e)
                {
                    Console.WriteLine("Длины векторов не совпадают");
                }

                break;
            }
            case "3":
            {
                LogVectors(vectors);

                int idx, value;

                do
                {
                    Console.Write("Введите индекс вектора: ");
                    inp = Console.ReadLine();
                } while (!int.TryParse(inp, out idx) || idx <= 0 || idx
> vectors.Count);

                do
                {
                    Console.Write("Введите значение на которое умножить
число: ");
                    inp = Console.ReadLine();
                } while (!int.TryParse(inp, out value));

                var result = Vectors.MultiplyByNumber(vectors[idx - 1],
value);
                result.Log($"Результат сложения умножения вектора на
число");
```

```
                    break;
                }
                case "4":
                {
                    LogVectors(vectors);

                    int idx;

                    do
                    {
                        Console.Write("Введите номер вектора для удаления:
");
                        inp = Console.ReadLine();
                    } while (!Int32.TryParse(inp, out idx) || idx < 1 || idx
> vectors.Count);

                    double norm = vectors[idx - 1].GetNorm();

                    Console.WriteLine($"Модуль вектора: {norm}");

                    break;
                }
                case "5":
                {
                    vectors.Add(GetVectorFromUserInput());
                    LogVectors(vectors);
                    break;
                }
                case "6":
                {
                    LogVectors(vectors);
                    int idx;

                    do
                    {
                        Console.Write("Введите номер вектора для удаления:
");
                        inp = Console.ReadLine();
                    } while (!Int32.TryParse(inp, out idx) || idx < 1 || idx
> vectors.Count);

                    vectors.Remove(vectors[idx - 1]);

                    LogVectors(vectors);
```

```
                    break;
                }
                case "7":
                {
                    LogVectors(vectors);
                    int idx;

                    do
                    {
                        Console.Write("Введите номер вектора для
клонирования: ");
                        inp = Console.ReadLine();
                    } while (!Int32.TryParse(inp, out idx) || idx < 1 || idx
> vectors.Count);

                    var vec = vectors[idx - 1];
                    IVectorable clone;
                    if (vec is ArrayVector)
                    {
                        clone = (vec as ArrayVector).Clone() as IVectorable;
                    }
                    else
                    {
                        clone = (vec as LinkedListVector).Clone() as
IVectorable;
                    }

                    vectors.Add(clone);

                    LogVectors(vectors);

                    break;
                }
                case "8":
                {
                    LogVectors(vectors);

                    try
                    {
                        int minLength = vectors[0].Length;
                        int maxLength = vectors[0].Length;

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

```
                {
                    if (minLength > vectors[i].Length)
                    {
                        minLength = vectors[i].Length;
                    }

                    if (maxLength < vectors[i].Length)
                    {
                        maxLength = vectors[i].Length;
                    }
                }

                string type;
                Console.WriteLine("Вектора с минимальным значением
координат: ");
                for (int i = 0; i < vectors.Count; i++)
                {
                    if (vectors[i].Length == minLength)
                    {
                        type = vectors[i] is ArrayVector ?
"ArrayVector" : "LinkedListVector";
                        Console.WriteLine($"{type}:
{vectors[i].ToString()}");
                    }
                }

                Console.WriteLine("Вектора с максимальным значением
координат: ");
                for (int i = 0; i < vectors.Count; i++)
                {
                    if (vectors[i].Length == maxLength)
                    {
                        type = vectors[i] is ArrayVector ?
"ArrayVector" : "LinkedListVector";
                        Console.WriteLine($"{type}:
{vectors[i].ToString()}");
                    }
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("Нет векторов");
            }
```

```
                    break;
                }
                case "9":
                {
                    IVectorable tmp;
                    VectorsComparer comparer = new VectorsComparer();

                    for (int i = 0; i < vectors.Count; i++)
                    {
                        for (int j = 0; j < vectors.Count - i - 1; j++)
                        {
                            int compareResult;
                            if (vectors[i] is ArrayVector)
                            {
                                compareResult      =      (vectors[i]      as
ArrayVector).CompareTo(vectors[j + 1]);
                            }
                            else
                            {
                                compareResult      =      (vectors[i]      as
LinkedListVector).CompareTo(vectors[j + 1]);
                            }

                            if (compareResult > 0)
                            {
                                tmp = vectors[j];
                                vectors[j] = vectors[j + 1];
                                vectors[j + 1] = tmp;
                            }
                        }
                    }

                    Console.WriteLine("Список  векторов  после  сортировки  по
длине: ");
                    LogVectors(vectors);

                    break;
                }
                case "10":
                {
                    IVectorable tmp;
                    VectorsComparer comparer = new VectorsComparer();

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

```
                    {
                        for (int j = 0; j < vectors.Count - i - 1; j++)
                        {
                            if (comparer.Compare(vectors[j], vectors[i + 1])
> 0)
                            {
                                tmp = vectors[j];
                                vectors[j] = vectors[j + 1];
                                vectors[j + 1] = tmp;
                            }
                        }
                    }

                    Console.WriteLine("Список векторов после сортировки по
модулю: ");
                    LogVectors(vectors);

                    break;
                }
                default:
                    Console.WriteLine("Нет такого пункта в меню");
                    break;
            }
        }
    }

    public static IVectorable GetVectorFromUserInput()
    {
        IVectorable vec;
        string inp;
        do
        {
            Console.WriteLine("Выберете тип вектора:\n\n" +
                              "\t1 - Вектор\n" +
                              "\t2 - Связный список\n");
            inp = Console.ReadLine();
        } while (inp != "1" && inp != "2");

        if (inp == "1")
        {
            vec = ArrayVector.GetFromUserInput();
        }
        else
        {
```

```
            int length;
            do
            {
                Console.Write("Введите длину связного списка: ");
                inp = Console.ReadLine();
            } while (!Int32.TryParse(inp, out length));

            vec = new LinkedListVector(length);
        }

        return vec;
    }

    public static void LogVectors(List<IVectorable> vectors)
    {
        for (int i = 0; i < vectors.Count; ++i)
        {
            IVectorable vec = vectors[i];

            string typeView = vec is ArrayVector ? "ArrayVector" :
"LinkedListVector";

            vec.Log($"{i + 1}: {typeView}");
        }
    }
}
namespace Lab04;

public class VectorsComparer : IComparer<IVectorable>
{
    public int Compare(IVectorable a, IVectorable b)
    {
        if (a.GetNorm() < b.GetNorm()) return -1;
        if (a.GetNorm() > b.GetNorm()) return 1;
        return 0;
    }
}
namespace Lab04;

public class Vectors
{
    public static IVectorable Sum(IVectorable a, IVectorable b)
    {
        if (a.Length != b.Length)
```

```
        {
            throw new Exception("Vectors norms are not equals");
        }

        IVectorable vec = new ArrayVector(a.Length);
        for (int i = 0; i < vec.Length; i++)
        {
            vec[i] = a[i] + b[i];
        }

        return vec;
    }

    public static double ScalarMultiply(IVectorable a, IVectorable b)
    {
        if (a.Length != b.Length)
        {
            throw new Exception("Vectors norms are not equal");
        }

        int result = 0;
        for (int i = 0; i < a.Length; i++)
        {
            result += a[i] * b[i];
        }

        return result;
    }

    public static IVectorable MultiplyByNumber(IVectorable vector, int
number)
    {
        for (int i = 0; i < vector.Length; i++)
        {
            vector[i] *= number;
        }

        return vector;
    }

    public static double GetNormSt(IVectorable vector)
    {
        return vector.GetNorm();
    }
```

```
}
namespace Lab04;

public class LinkedListVector : IVectorable
{
    private Node head;

    private class Node
    {
        public int value = 0;
        public Node next = null;

        public Node(int value)
        {
            this.value = value;
            next = null;
        }
    }

    public LinkedListVector()
    {
        var r = new Random();

        head = new Node(r.Next(100));
        Node cur = head;

        for (int i = 0; i < 5; i++)
        {
            cur.next = new Node(r.Next(100));
            cur = cur.next;
        }
    }

    public LinkedListVector(int length)
    {
        var r = new Random();

        head = new Node(r.Next(100));
        Node cur = head;

        for (int i = 0; i < length; i++)
        {
            cur.next = new Node(r.Next(100));
            cur = cur.next;
```

```
        }
    }

    public int this[int idx]
    {
        get
        {
            if (0 <= idx && idx <= Length)
            {
                Node cur = head;
                for (int i = 0; i < idx; i++)
                {
                    cur = cur.next;
                }

                return cur.value;
            }
            else
            {
                throw new IndexOutOfRangeException("Индекс за границами
связного списка");
            }
        }
        set
        {
            if (0 <= idx && idx <= Length)
            {
                Node cur = head;
                for (int i = 0; i < idx; i++)
                {
                    cur = cur.next;
                }

                cur.value = value;
            }
            else
            {
                throw new IndexOutOfRangeException("Индекс за границами
связного списка");
            }
        }
    }

    public int Length
```

```
    {
        get
        {
            if (head == null)
            {
                return -1;
            }

            int length = 0;
            Node cur = head;
            while (cur.next != null)
            {
                cur = cur.next;
                length++;
            }

            return length;
        }
    }

    public double GetNorm()
    {
        double acc = 0;
        Node cur = head;
        for (int i = 0; i < Length; i++)
        {
            acc += Math.Pow(cur.value, 2);
            cur = cur.next;
        }

        return Math.Sqrt(acc);
    }

    public void InsertByIndex(int idx, int value)
    {
        if (idx < 0 || idx > Length) throw new
IndexOutOfRangeException("Индекс за границами связного списка");

        Node node = new Node(value);

        if (idx == 0) {
            node.next = head;
            head = node;
            return;
```

```
        }

        Node cur = head;
        int curIndex = 0;
        while (cur != null && curIndex < idx - 1) {
            cur = cur.next;
            curIndex++;
        }

        if (cur == null) throw new IndexOutOfRangeException("Индекс за границами связного списка");

        node.next = cur.next;
        cur.next = node;
    }

    public void InsertToStart(int value)
    {
        InsertByIndex(0, value);
    }

    public void InsertToEnd(int value)
    {
        InsertByIndex(Length, value);
    }

    public void DeleteByIndex(int idx)
    {
        if (head == null) throw new Exception("Связный список пуст");
        if (idx < 0 || idx >= Length) throw new IndexOutOfRangeException("Индекс за границами связного списка");

        Node cur = head;

        if (idx == 0)
        {
            head = cur.next;
            return;
        }

        for (int i = 0; cur != null && i < idx - 1; i++)
        {
            cur = cur.next;
        }
```

```
        if (cur == null || cur.next == null) return;

        cur.next = cur.next.next;
    }

    public void DeleteFromStart()
    {
        DeleteByIndex(0);
    }

    public void DeleteFromEnd()
    {
        DeleteByIndex(Length - 1);
    }

    public void Log(string message = "")
    {
        if (message != "") Console.Write($"{message}: ");

        Console.WriteLine(ToString());
    }

    public override string ToString()
    {
        string s = Length.ToString() + ' ';

        for (int i = 0; i < Length; i++)
        {
            s += this[i].ToString();
            if (i != Length - 1) s += ' ';
        }
        return s;
    }

    public int CompareTo(object? obj)
    {
        if (!(obj is IVectorable))
        {
            throw new Exception("Можно сравнить только объекты типа
IVectorable");
        }

        IVectorable other = obj as IVectorable;
```

```
        if (Length < other.Length) return -1;
        if (Length > other.Length) return 1;
        return 0;
    }

    public override bool Equals(object? obj)
    {
        if (!(obj is IVectorable))
        {
            throw new Exception("Можно сравнивать только объекты типа
IVectorable");
        }

        IVectorable other = obj as IVectorable;

        if (Length != other.Length) return false;

        for (int i = 0; i < Length; i++)
        {
            if (this[i] != other[i]) return false;
        }

        return true;
    }

    public object Clone()
    {
        LinkedListVector clone = new LinkedListVector(Length);

        for (int i = 0; i < Length; i++)
        {
            clone[i] = this[i];
        }

        return clone;
    }
}
```

```
Выберете действие:

        1  - Сумма векторов
        2  - Скалярное умножение
        3  - Умножение на число
        4  - Рассчитать модуль вектора
        5  - Добавить вектор в список
        6  - Удалить вектор из списка
        7  - Клонировать вектор
        8  - Вывести вектора с минимальным и максимальным количеством координат
        9  - Отсортировать вектора по количеству координат
        10 - Отсортировать вектора по модулю
        11 - Сравнить вектора
        0  - Выход
```

Рисунок 1 – Главное меню программы

```
1: LinkedListVector: 5 75 40 26 39 92
2: LinkedListVector: 5 93 53 70 80 51
Выберете действие:

        1  - Сумма векторов
        2  - Скалярное умножение
        3  - Умножение на число
        4  - Рассчитать модуль вектора
        5  - Добавить вектор в список
        6  - Удалить вектор из списка
        7  - Клонировать вектор
        8  - Вывести вектора с минимальным и максимальным количеством координат
        9  - Отсортировать вектора по количеству координат
        10 - Отсортировать вектора по модулю
        11 - Сравнить вектора
        0  - Выход

11
Введите индекс первого вектора: 1
Введите индекс второго вектора: 2
Вектора не равны
```

Рисунок 2 – Проверка векторов на равенства

```
1: LinkedListVector: 5 75 40 26 39 92
2: LinkedListVector: 5 93 53 70 80 51
3: LinkedListVector: 1 61
4: LinkedListVector: 2 64 63
5: LinkedListVector: 8 57 75 65 20 98 47 26 89
6: LinkedListVector: 17 48 94 3 54 98 88 29 49 59 79 77 55 26 4 99 21 80
Вектора с минимальным значением координат:
LinkedListVector: 1 61
Вектора с максимальным значением координат:
LinkedListVector: 17 48 94 3 54 98 88 29 49 59 79 77 55 26 4 99 21 80
```

Рисунок 3 – Вектора с минимальным и максимальным количеством координат

```
Список векторов после сортировки по длине:
1: LinkedListVector: 1 78
2: LinkedListVector: 2 8 44
3: LinkedListVector: 3 42 96 7
4: LinkedListVector: 7 22 86 2 0 47 27 23
```

Рисунок 4 – Отсортированные по количеству координат вектора

# ВЫВОДЫ

В лабораторной работе были использованы конструкции языка:

- форматированный вывод информации на консоль;
- оператор switch;
- условные операторы;
- функции;
- классы;
- конструкторы класса;
- поля класса;
- статические и динамические методы класса;
- интерфейсы;
- индексаторы;
- конструкция try-catch.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Павловская Т.А. C#. Программирование на языке высокого уровня. Учебник для вузов [Текст]/Т.А. Павловская. – СПб.: Питер, 2007. – 432 с.