Защищено:
Гапанюк Ю.Е.

"__"_____**2023** г.

Демонстрация:
Казакова В.В.

"__"_____**2023** г.

Отчет по лабораторной работе № 3 по курсу Парадигмы и конструкции языков программирования

Тема работы: " Работа с коллекциями"

7
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5Ц-51Б _____
(подпись)

Казакова В.В. "__"_____**2023** г.

Москва, МГТУ - **2023**

1. Описание задания

1. Программа должна быть разработана в виде консольного приложения на языке C#.

2. Создать объекты классов «Прямоугольник», «Квадрат», «Круг».

3. Для реализации возможности сортировки геометрических фигур для класса «Геометрическая фигура» добавить реализацию интерфейса IComparable. Сортировка производится по площади фигуры.

4. Создать коллекцию класса ArrayList. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.

5. Создать коллекцию класса List<Figure>. Сохранить объекты в коллекцию. Отсортировать коллекцию. Вывести в цикле содержимое коллекции.

6. Модифицировать класс разреженной матрицы (проект SparseMatrix) для работы с тремя измерениями – x,y,z. Вывод элементов в методе ToString() осуществлять в том виде, который Вы считаете наиболее удобным. Разработать пример использования разреженной матрицы для геометрических фигур.

7. Реализовать класс «SimpleStack» на основе односвязного списка. Класс SimpleStack наследуется от класса SimpleList (проект SimpleListProject). Необходимо добавить в класс методы:

• public void Push(T element) – добавление в стек;

• public T Pop() – чтение с удалением из стека.

8. Пример работы класса SimpleStack реализовать на основе геометрических фигур.

2. Текст программы

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

class Program
{
    static void Main()
    {
```

```
            // Создание объектов классов
            Rectangle rectangle = new Rectangle(4, 5);
            Square square = new Square(4);
            Circle circle = new Circle(3);

            // Создание коллекции ArrayList
            ArrayList figuresArrayList = new ArrayList
            {
                rectangle,
                square,
                circle
            };

            figuresArrayList.Sort(new FigureComparer());

            // Вывод содержимого коллекции ArrayList
            Console.WriteLine("ArrayList:");
            foreach (Figure figure in figuresArrayList)
            {
                Console.WriteLine($"Площадь: {figure.GetArea()}");
            }

            // Создание коллекции List<Figure>
            List<Figure> figuresList = new List<Figure>
            {
                rectangle,
                square,
                circle
            };

            figuresList.Sort(new FigureComparer());

            // Вывод содержимого коллекции List<Figure>
            Console.WriteLine("\nList<Figure>:");
            foreach (Figure figure in figuresList)
            {
                Console.WriteLine($"Площадь: {figure.GetArea()}");
            }

            // Пример использования разреженной матрицы для геометрических фигур
            SparseMatrix<Figure> figureMatrix = new SparseMatrix<Figure>(3, 3, 3);
            figureMatrix[0, 0, 0] = rectangle;
            figureMatrix[1, 1, 1] = square;
            figureMatrix[2, 2, 2] = circle;

            Console.WriteLine("\nРазреженная матрица:");
            Console.WriteLine(figureMatrix);

            // Пример работы класса SimpleStack
            SimpleStack<Figure> figureStack = new SimpleStack<Figure>();
            figureStack.Push(rectangle);
            figureStack.Push(square);
            figureStack.Push(circle);

            Console.WriteLine("\nSimpleStack:");
            while (!figureStack.IsEmpty)
            {
                Figure poppedFigure = figureStack.Pop();
                Console.WriteLine($"Площадь: {poppedFigure.GetArea()}");
                Console.ReadKey();
            }
        }
    }

    public abstract class Figure : IComparable<Figure>
    {
```

```
    public abstract double GetArea();

    public int CompareTo(Figure other)
    {
        if (other == null) return 1;
        return GetArea().CompareTo(other.GetArea());
    }
}

public class Rectangle : Figure
{
    public double Width { get; }
    public double Height { get; }

    public Rectangle(double width, double height)
    {
        Width = width;
        Height = height;
    }

    public override double GetArea()
    {
        return Width * Height;
    }
}

public class Square : Figure
{
    public double Side { get; }

    public Square(double side)
    {
        Side = side;
    }

    public override double GetArea()
    {
        return Side * Side;
    }
}

public class Circle : Figure
{
    public double Radius { get; }

    public Circle(double radius)
    {
        Radius = radius;
    }

    public override double GetArea()
    {
        return Math.PI * Radius * Radius;
    }
}

public class FigureComparer : IComparer<Figure>, IComparer
{
    public int Compare(Figure x, Figure y)
    {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        return x.CompareTo(y);
    }
```

```
    public int Compare(object x, object y)
    {
        if (x == null && y == null) return 0;
        if (x == null) return -1;
        if (y == null) return 1;

        if (x is Figure figureX && y is Figure figureY)
        {
            return Compare(figureX, figureY);
        }

        throw new ArgumentException("Both objects must be of type Figure.");
    }
}

public class SparseMatrix<T> where T : Figure
{
    private readonly Dictionary<(int x, int y, int z), T> matrix = new
Dictionary<(int x, int y, int z), T>();
    private readonly int maxX, maxY, maxZ;

    public SparseMatrix(int maxX, int maxY, int maxZ)
    {
        this.maxX = maxX;
        this.maxY = maxY;
        this.maxZ = maxZ;
    }

    public T this[int x, int y, int z]
    {
        get => matrix.TryGetValue((x, y, z), out T value) ? value :
default(T);
        set => matrix[(x, y, z)] = value;
    }

    public override string ToString()
    {
        string result = "";
        for (int z = 0; z < maxZ; z++)
        {
            for (int y = 0; y < maxY; y++)
            {
                for (int x = 0; x < maxX; x++)
                {
                    result += $"{this[x, y, z]?.GetArea() ?? 0:F2}\t";
                }
                result += "\n";
            }
            result += "\n";
        }
        return result;
    }
}

public class SimpleNode<T>
{
    public T Data { get; set; }
    public SimpleNode<T> Next { get; set; }

    public SimpleNode(T data)
    {
        Data = data;
    }
}
```

```
public class SimpleList<T>
{
    protected SimpleNode<T> head;

    public void Add(T data)
    {
        SimpleNode<T> node = new SimpleNode<T>(data);
        node.Next = head;
        head = node;
    }

    public bool IsEmpty => head == null;
}

public class SimpleStack<T> : SimpleList<T>
{
    public void Push(T element)
    {
        Add(element);
    }

    public T Pop()
    {
        if (IsEmpty)
        {
            throw new InvalidOperationException("The stack is empty.");
        }

        T poppedData = head.Data;
        head = head.Next;
        return poppedData;
    }
}
```

3. Экранные формы